IN4080 – 2020 FALL

NATURAL LANGUAGE PROCESSING

Jan Tore Lønning

Neural networks, Language models, word2wec

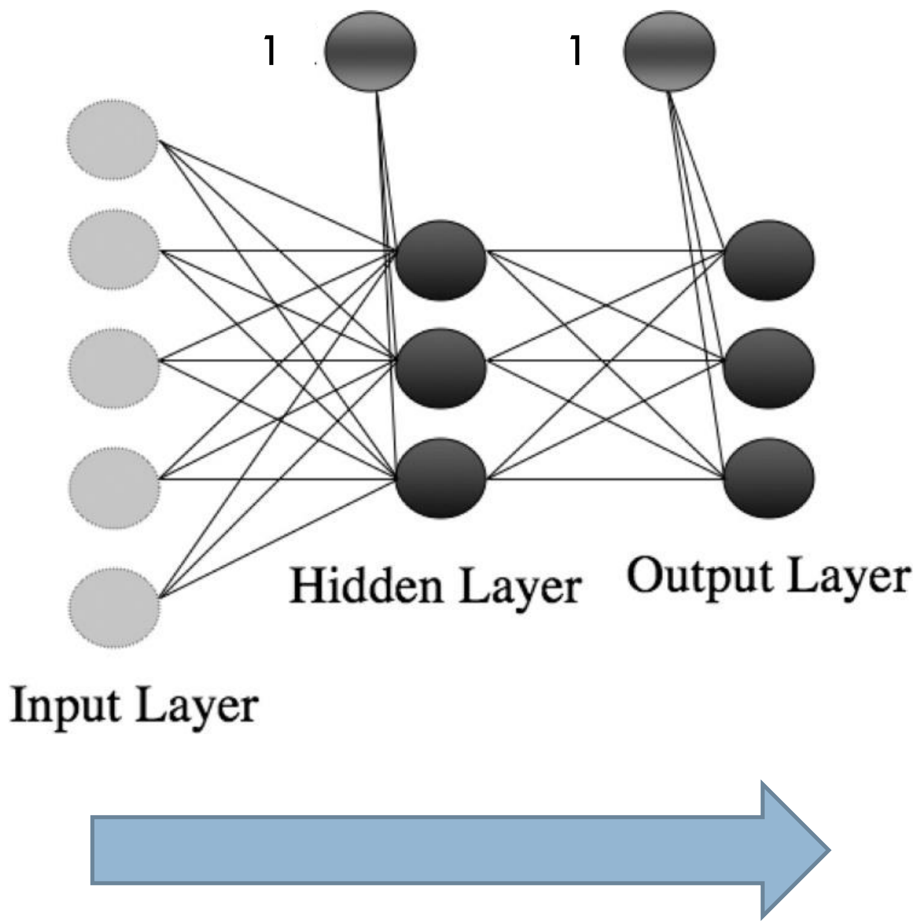
Lecture 6, 21 Sept

Today

- Neural networks
- Language models
- Word embeddings
- Word2vec

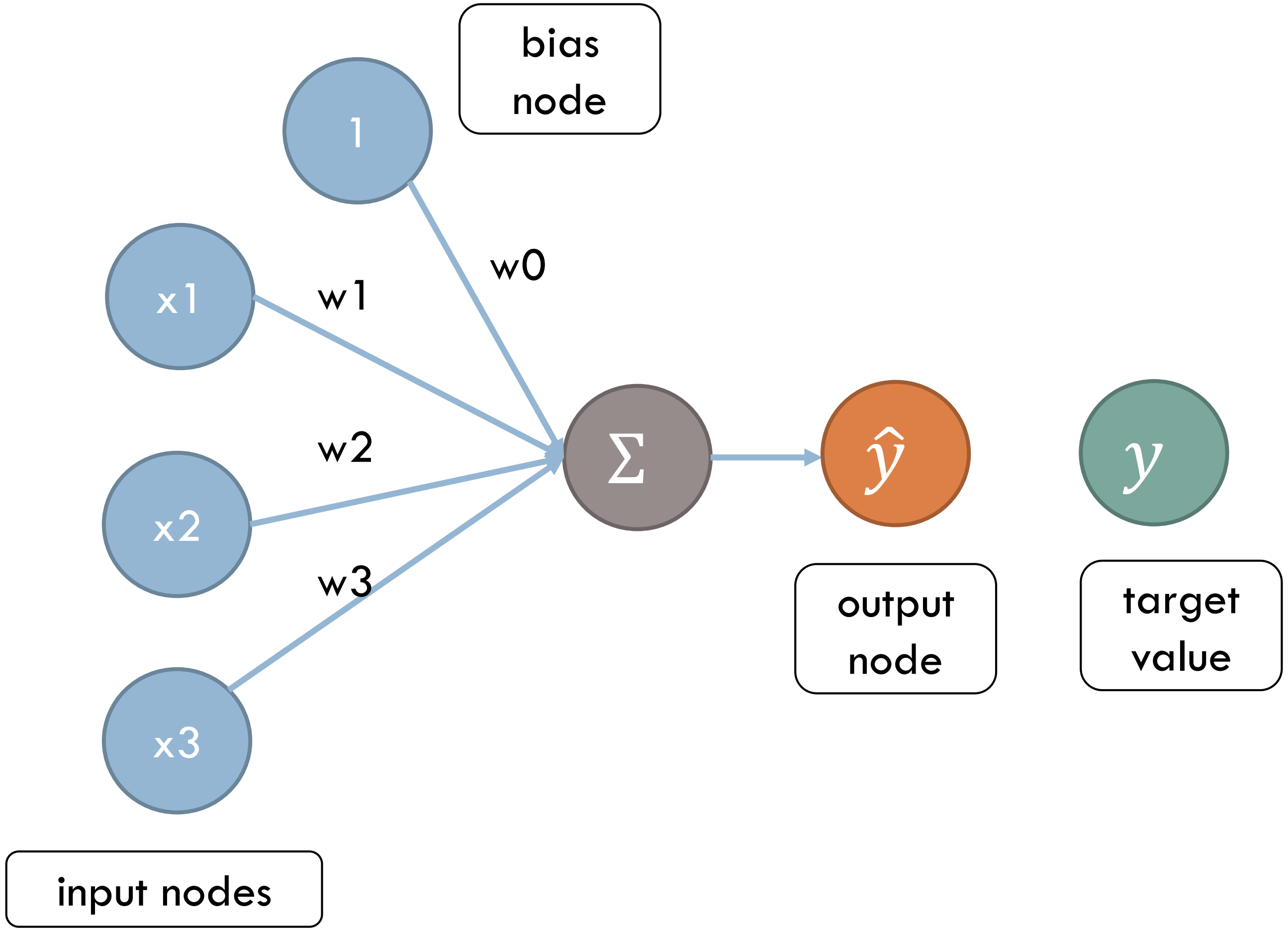
Artificial neural networks

- Inspired by the brain
  - neurons, synapses
- Does not pretend to be a model of the brain
- The simplest model is the
  - Feed forward network, also called
  - Multi-layer Perceptron



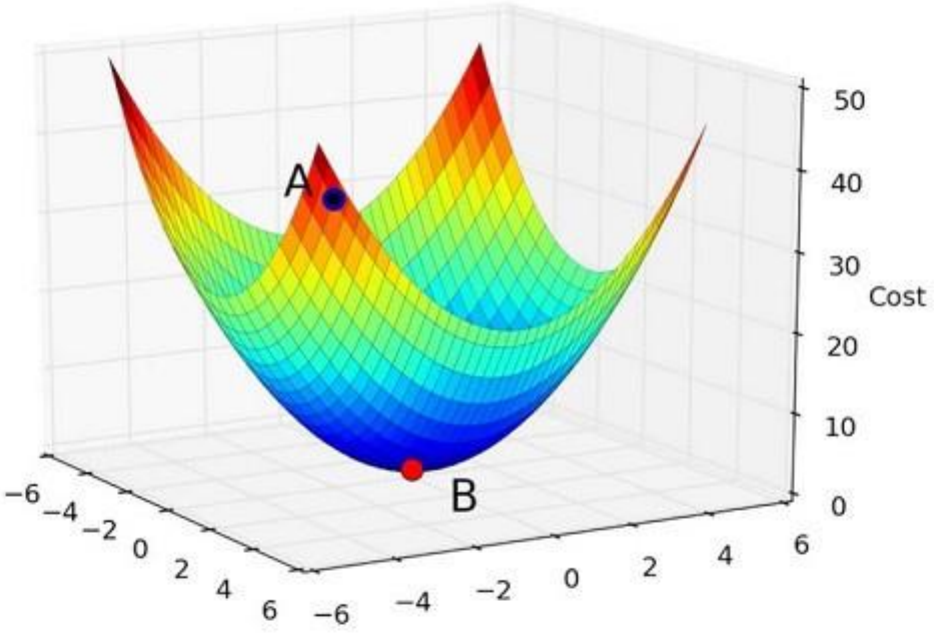
Linear regression as a network

- Each feature, x_i, of the input vector is an input node
- An additional bias node $x_0 = 1$ for the intercept
- A weight at each edge,
- Multiply the input values with the respective weights: $w_i x_i$
- Sum them
- $\hat{y} = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$



Gradient descent (for linear regression)

- We start with an initial set of weights
- Consider training examples
- Adjust the weights to reduce the loss
- How?
- Gradient descent
- Gradient means partial derivatives.



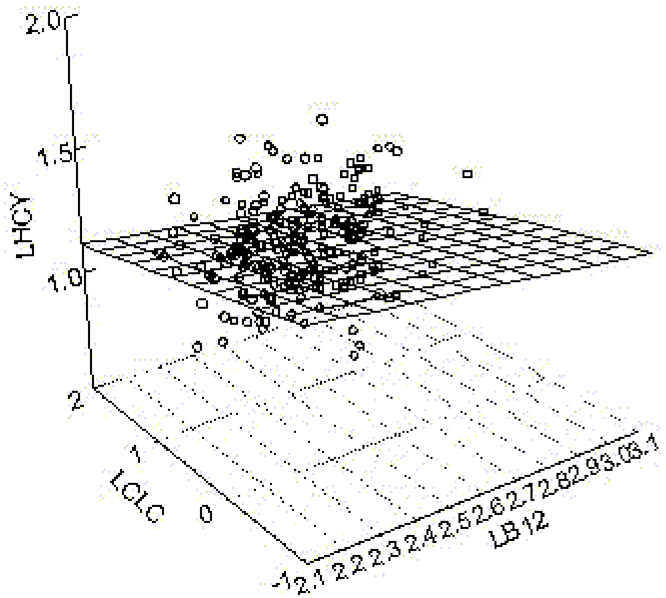
Linear regression: higher dimensions

- Linear regression of more than two variables works similarly
- We try to fit the best (hyper-)plane

$$\hat{y} = f(x_0, x_1, \dots, x_n) = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$

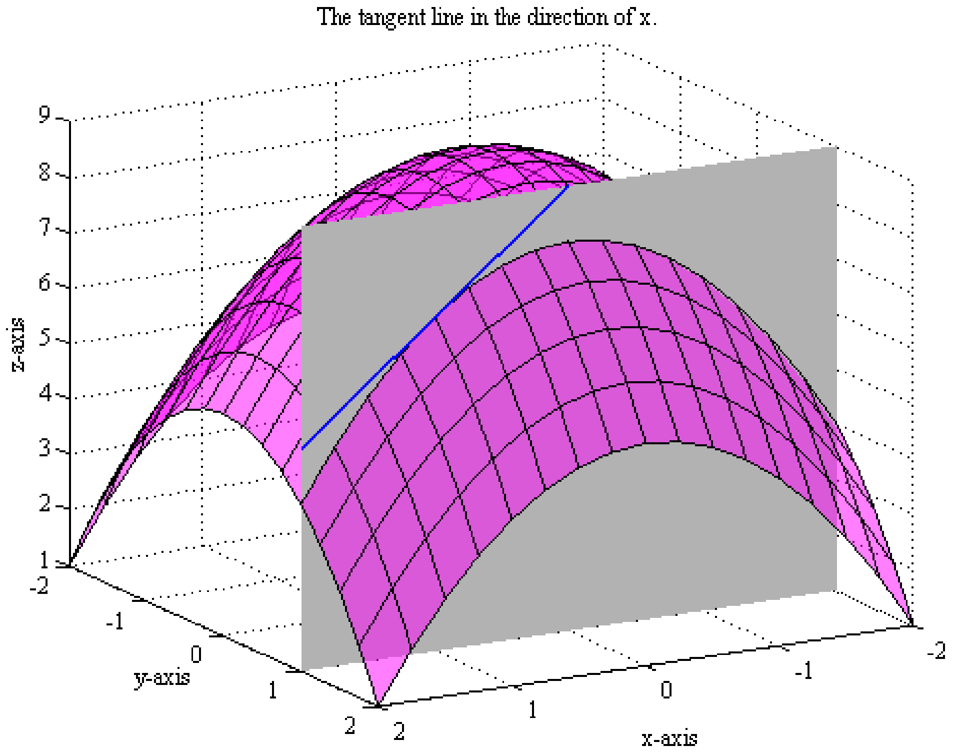
- We can use the same mean square:

$$\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$



Partial derivatives

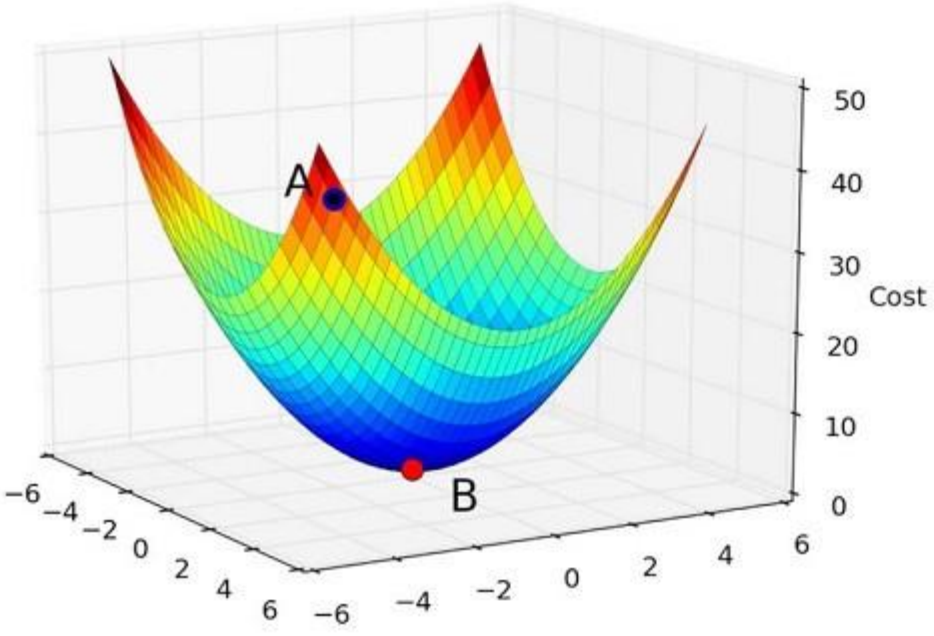
- A function of more than one variable, e.g. $f(x, y)$
- The partial derivative, e.g. $\frac{\partial f}{\partial x}$ is the derivative one gets by keeping the other variables constant
- E.g. if $f(x, y) = ax + by + c$, $\frac{\partial f}{\partial x} = a$ and $\frac{\partial f}{\partial y} = b$



https://www.wikihow.com/Image:OyXsh.png

Gradient descent

- We move in the opposite direction of where the gradient is pointing.
- Intuitively:
  - Take small steps in all direction parallel to the (feature) axes
  - The length of the steps are proportional to the steepness in each direction



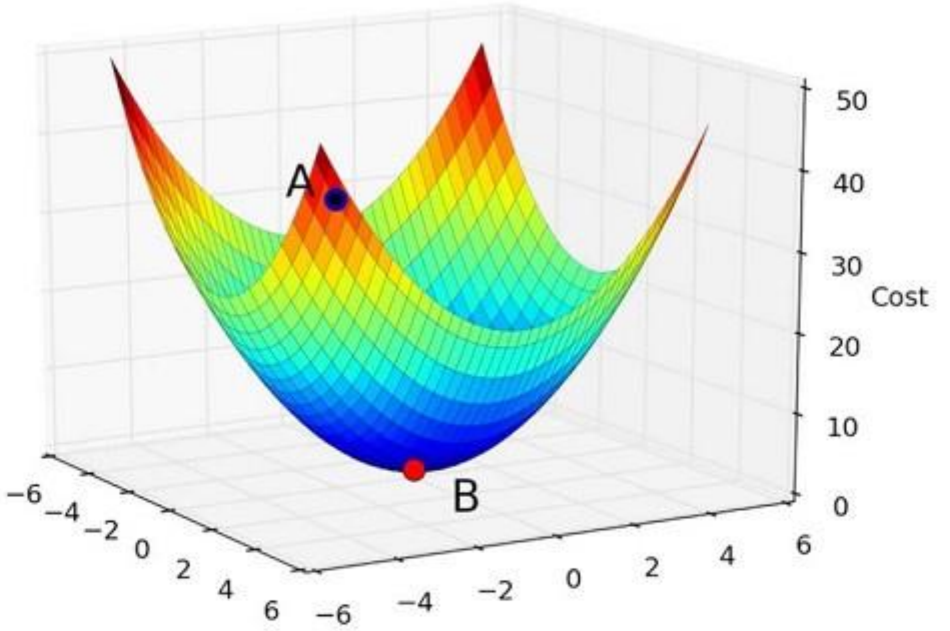
Properties of the derivatives

1. If $f(x) = ax + b$ then $f'(x) = a$
   - we also write $\frac{df}{dx} = a$
   - and if $y = f(x)$, we can write $\frac{dy}{dx} = a$
2. If $f(x) = x^n$ for an integer $\neq 0$ then $f'(x) = nx^{(n-1)}$
3. If $f(x) = g(y)$ and $y = h(x)$ then $f'(x) = g'(y)h'(x)$
   - if $z = f(x) = g(y)$, this can be written $\frac{dz}{dx} = \frac{dz}{dy}\ \frac{dy}{dx}$

- In particular, if $f(x) = (ax + b)^2$ then $f'(x) = 2(ax + b)a$

Gradient descent (for linear regression)

- Loss: Mean squared error :
  - $L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{j=1}^{n}\left(\hat{y}_j - y_j\right)^2$
  - $\hat{y}_j = \sum_{i=0}^{m} w_i x_{j,i} = \boldsymbol{w} \cdot \boldsymbol{x}_j$
- We will update the w_i-s
- Consider the partial derivatives w.r.t the w_i-s
- $\frac{\partial}{\partial w_i} L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{j=1}^{n} 2\left(\hat{y}_j - y_j\right) x_{j,i}$
- Update w_i: $w_i = w_i - \eta \frac{\partial}{\partial w_i} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$



n is the number of observations,
$0 \le j \le n$ and
m is the number of features for each observation,
$0 \le i \le m$

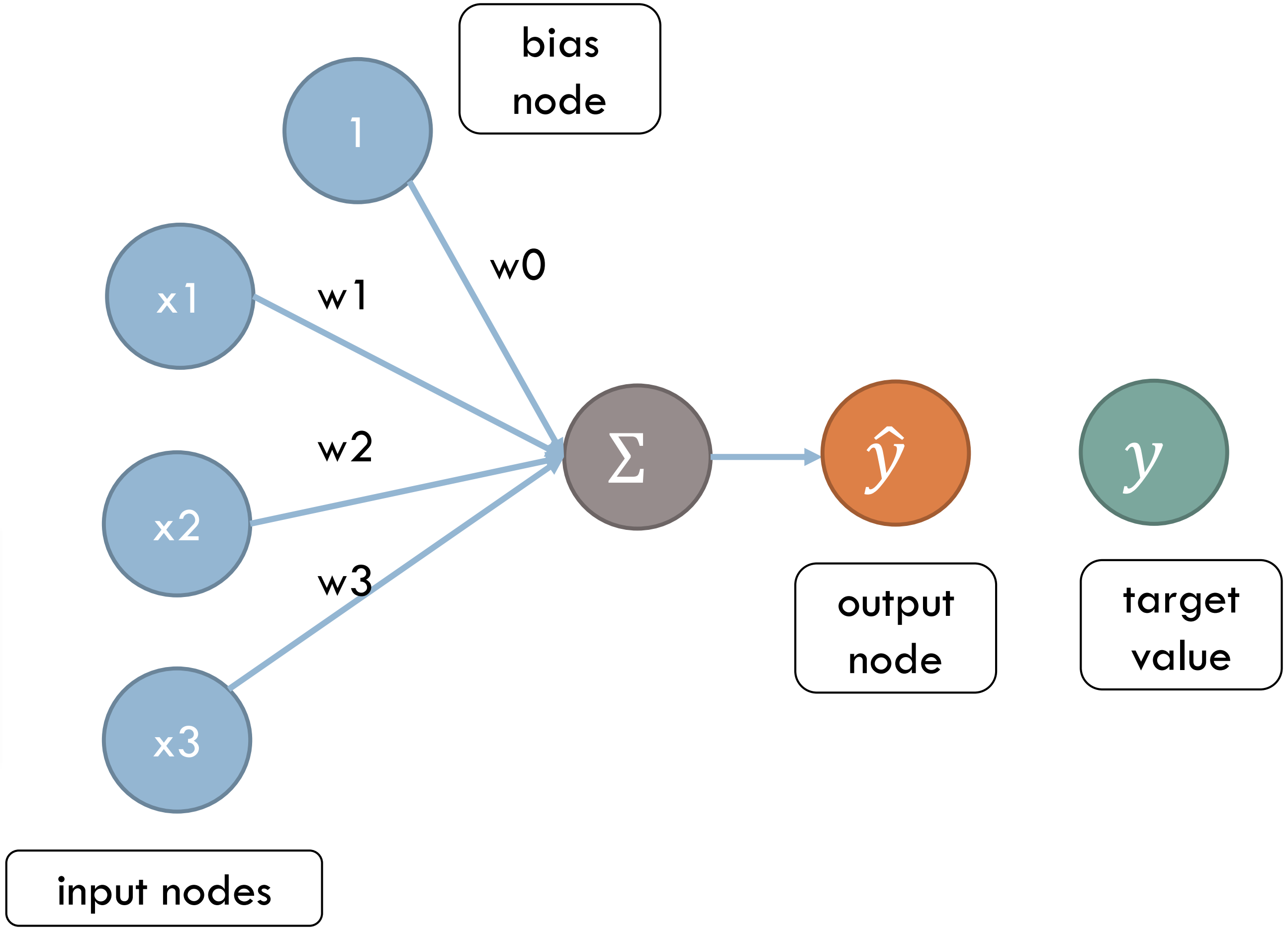
Inspecting the update

$$w_i = w_i - \eta \frac{1}{n} \sum_{j=1}^{n} 2\left(\hat{y}_j - y_j\right) x_{j,i}$$

The error term (delta term) of this prediction, from the loss function

The contribution to the error from this weight

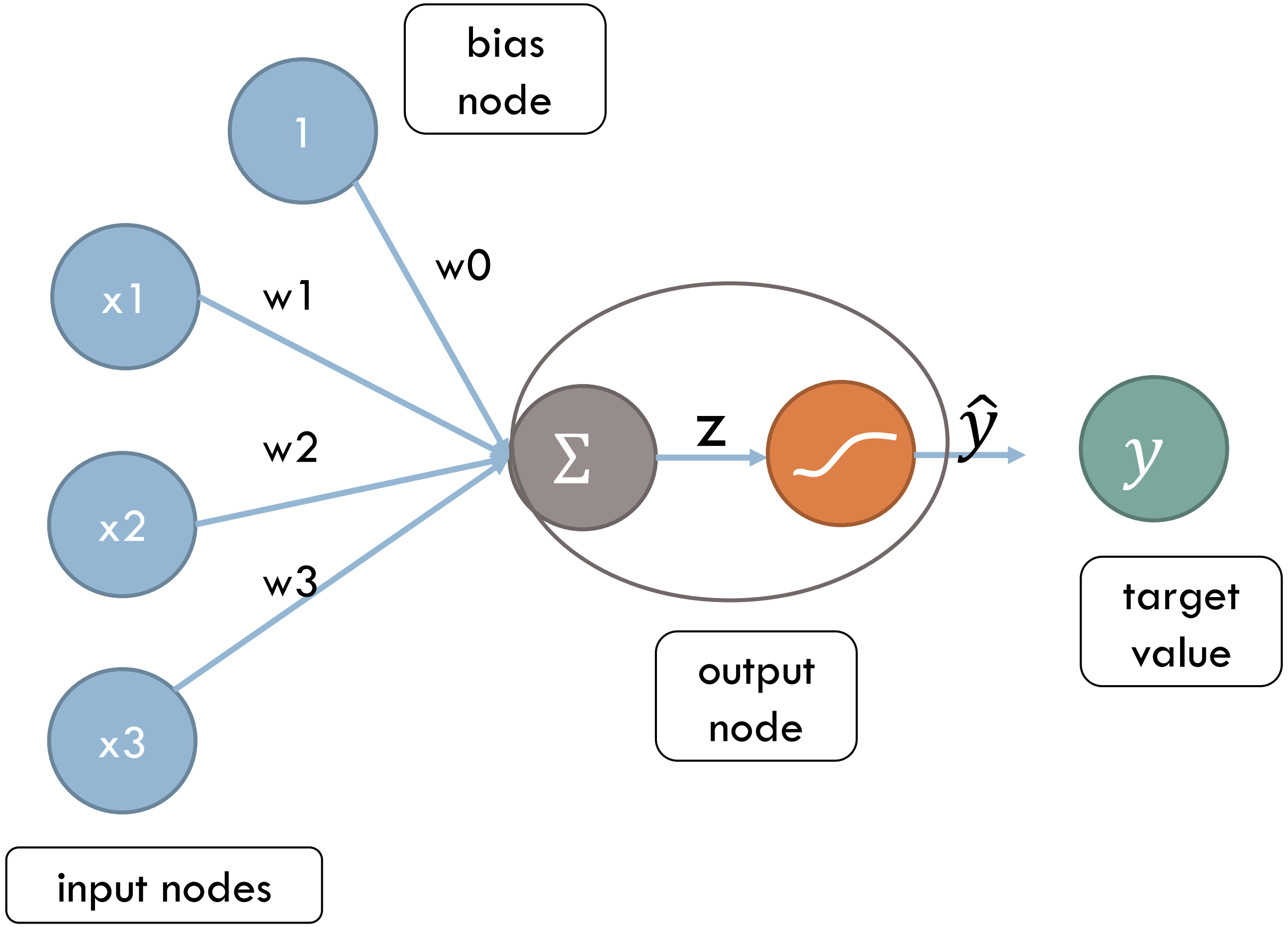
η is the learning rate



Logistic regression as a network

- $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
- $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$
- Loss: $L_{CE} = -\sum_{j=1}^{n} \log\left[\hat{y}_j{}^{j}(1-\hat{y}_j)^{(1-y_j)}\right]$
- $\frac{\partial}{\partial \widehat{w_i}} L_{CE} = \frac{\partial}{\partial \hat{y}} L_{CE} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$
- $\frac{\partial}{\partial \hat{y}} L_{CE} = \frac{(y-\hat{y})}{\hat{y}(1-\hat{y})}$
- $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$
- $\frac{\partial z}{\partial w_i} = x_i$
- $\frac{\partial}{\partial \widehat{w_i}} L_{CE} = \frac{(y-\hat{y})}{\hat{y}(1-\hat{y})} \hat{y}(1-\hat{y}) x_i = (y-\hat{y}) x_i$

To simplify, consider only one observation, y_j



Logistic regression as a network

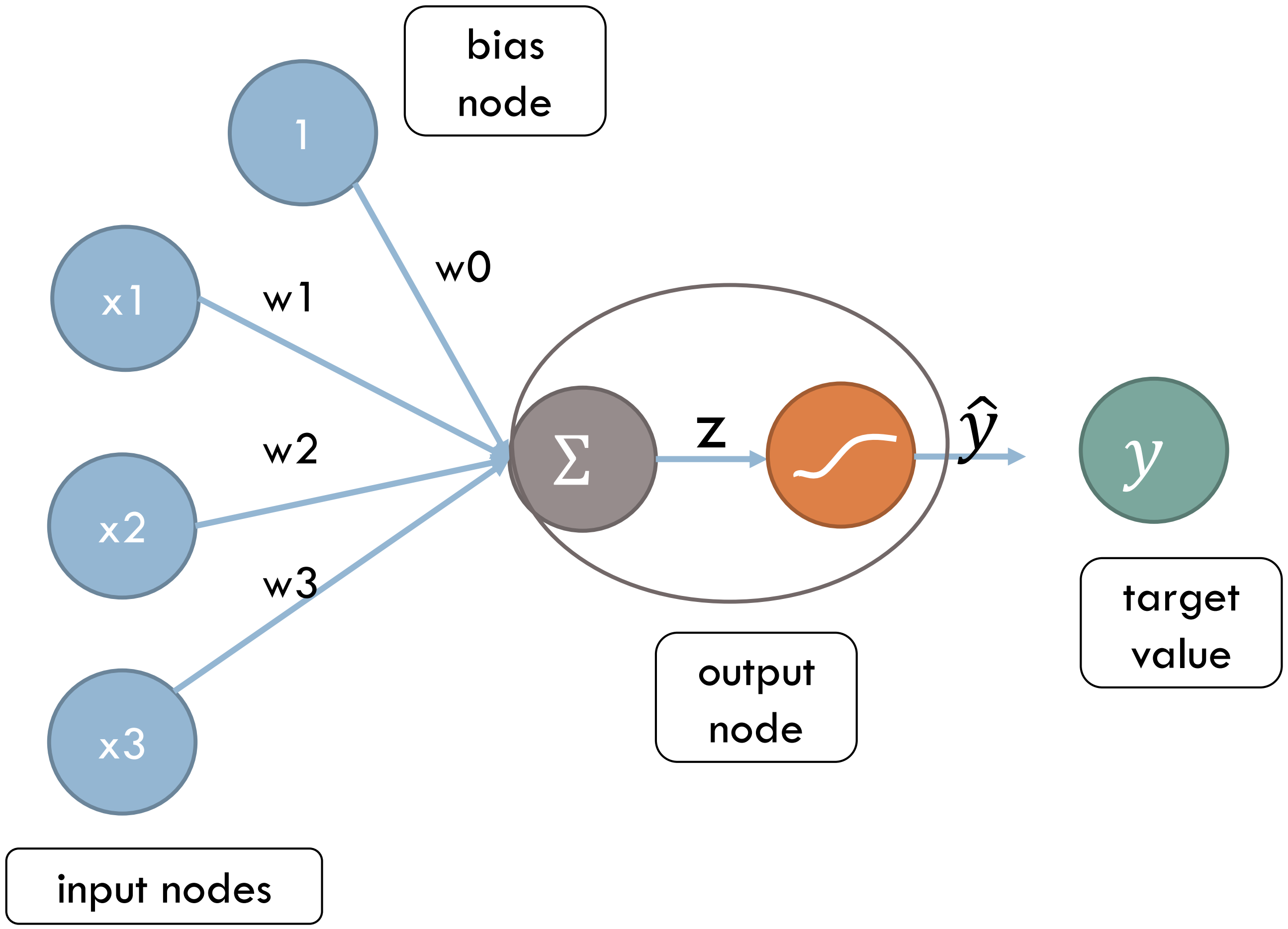
From the loss

From the activation function

$$\frac{\partial}{\partial \widehat{w_i}} L_{CE} = \frac{(y-\hat{y})}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})x_i = (y-\hat{y})x_i$$

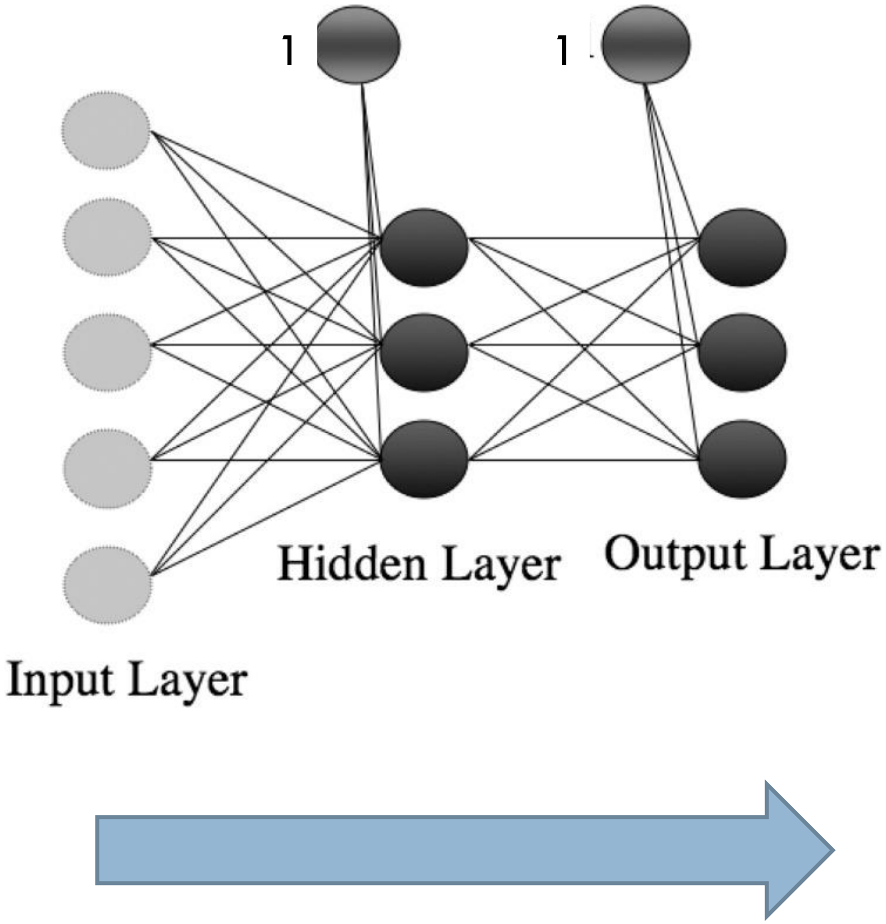
The delta term at the end of W

The contribution to the error from this weight



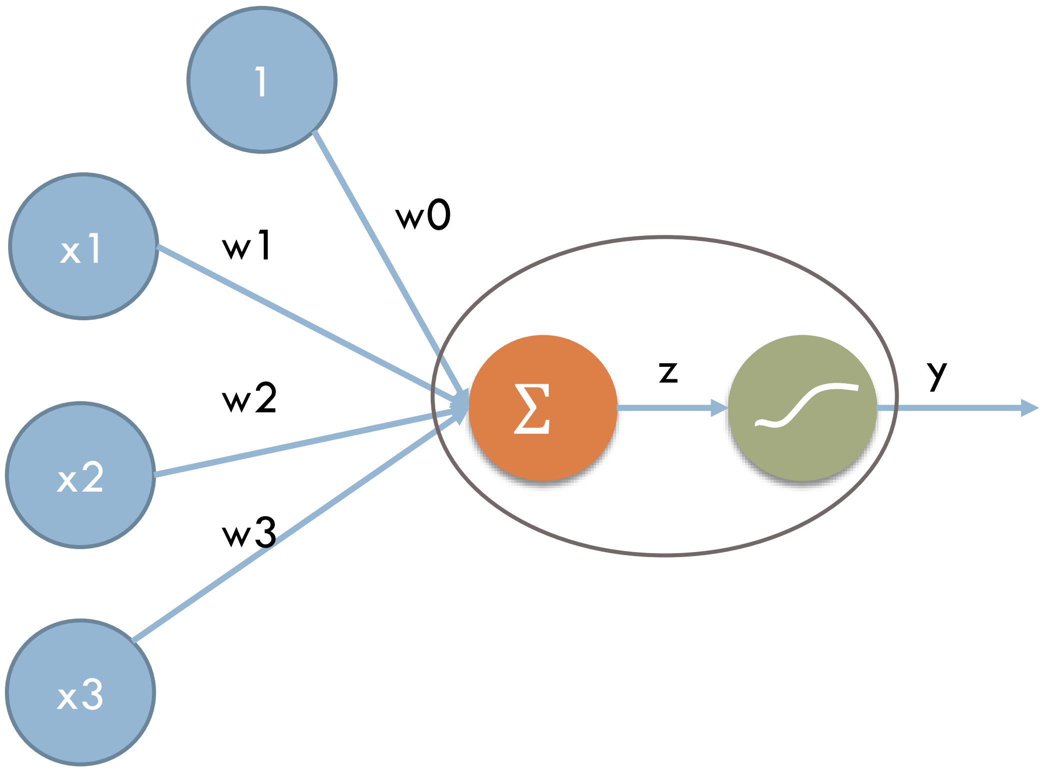
Feed forward network

- An input layer
- An output layer: the predictions
- One or more hidden layers
- Connections from one layer to the next (from left to right)



The hidden nodes

- Each hidden node is like a small logistic regression:
  - First sum of weighted inputs :
  - $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
  - Then the result is run through an activation function, e.g. σ
  - $y = \sigma(z) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$

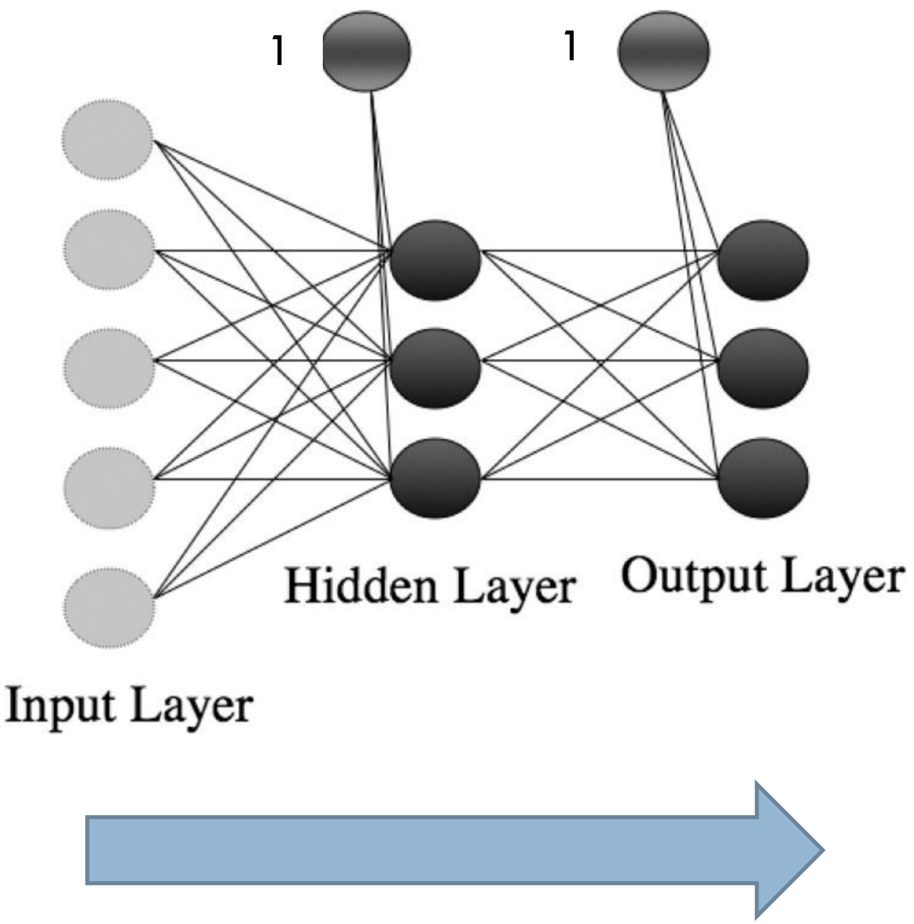


It is the non-linearity of the activation function which makes it possible for MLP to predict non-linear decision boundaries

The output layer

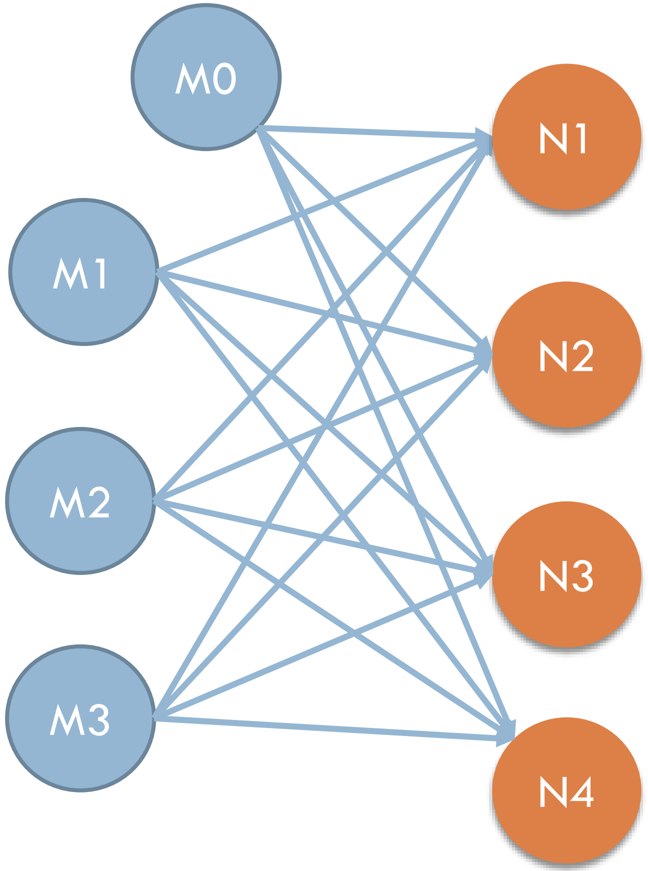
Alternatives

- Regression:
  - One node
  - No activation function
- Binary classifier:
  - One node
  - Logistic activation function
- Multinomial classifier
  - Several nodes
  - Softmax
- + more alternatives
- Choice of loss function depends on task



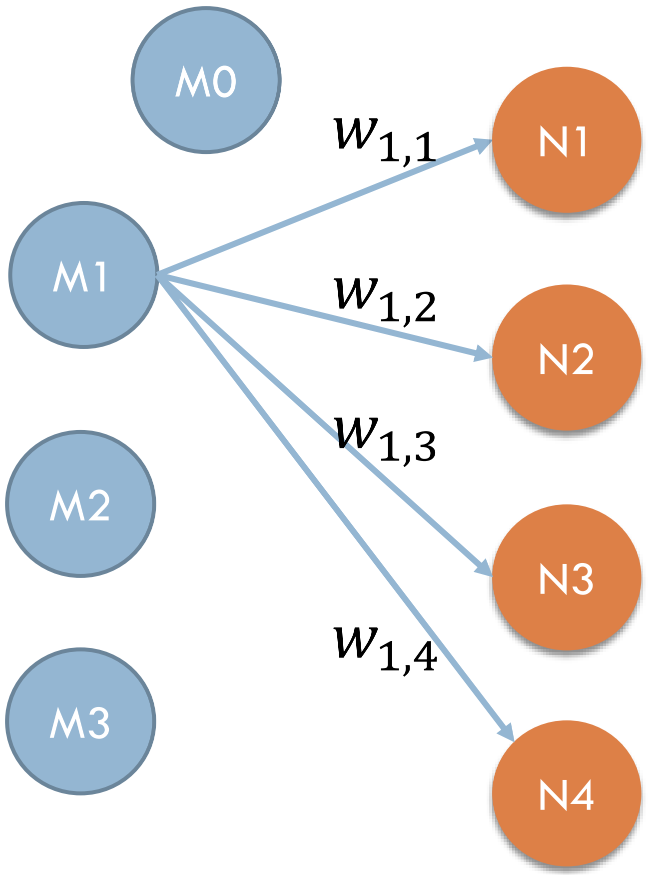
Learning in multi-layer networks

- Consider two consecutive layers:
  - Layer M, with $1 \leq i \leq m$ nodes, and a bias node M0
  - Layer N, with $1 \leq j \leq n$ nodes
  - Let $w_{i,j}$ be the weight at the edge going from M_i to N_j
- Consider processing one observation:
  - Let x_i be the value going out of node M_i
  - If M is a hidden layer:
    - $x_i = \sigma(z_i)$, where $z_i = \sum(\dots)$



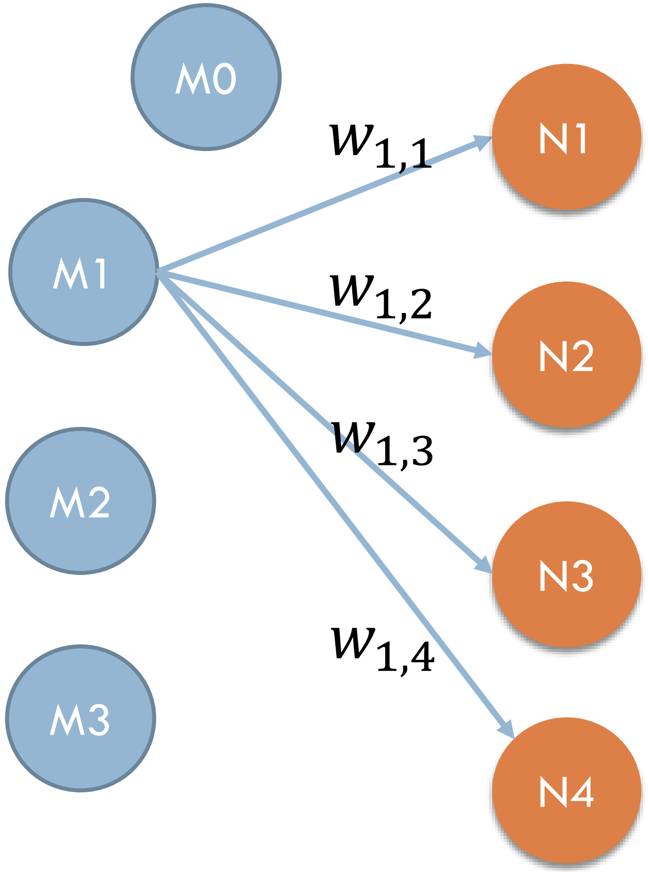
Learning in multi-layer networks

- If N is the output layer, calculate the error terms δ_j^N as before from the loss and the activation function at each node N_j
- If M is a hidden layer: Calculate the error term at the nodes combining
  - A weighted sum of the error terms at layer N
  - The derivative of the activation function
  - $\delta_i^M = \left(\sum_{j=1}^{n} w_{i,j}\delta_j^N\right)\frac{dx_i}{dz_i}$
    - where $x_i = \sigma(z_i)$, where $z_i = \sum(\ldots)$

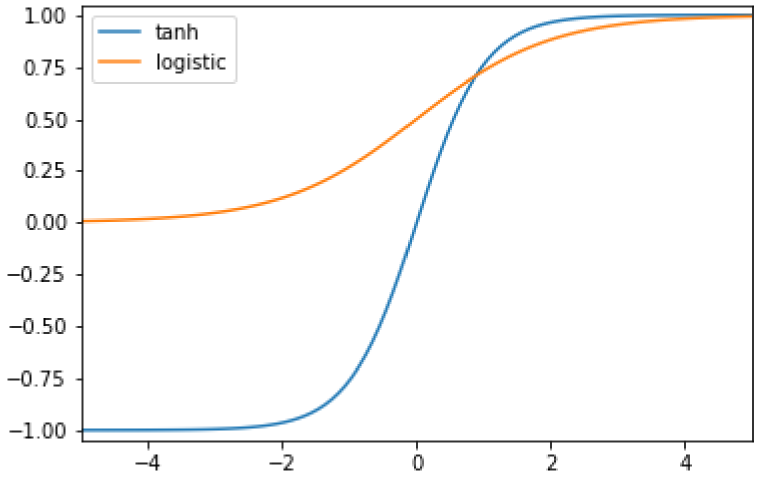


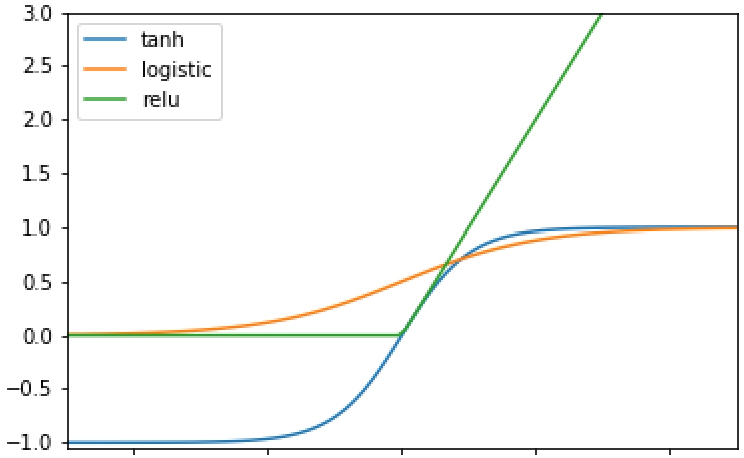
Learning in multi-layer networks

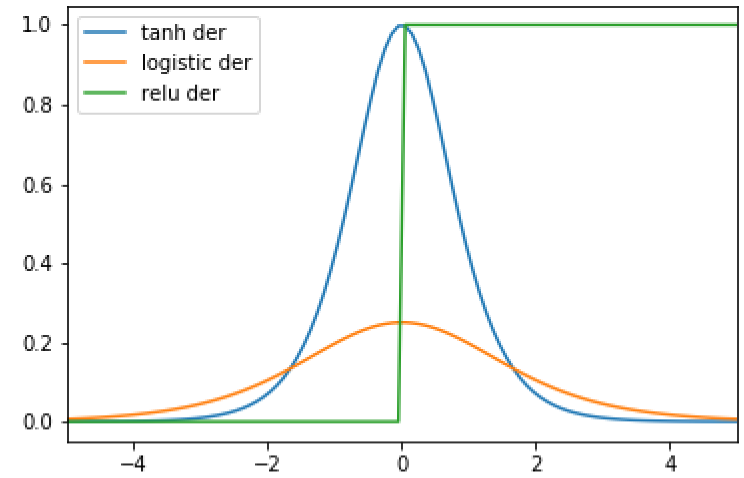
- By repeating the process, we get error terms at all nodes in all the hidden layers.
- The update of the weights between the layers can be done as before:
- $w_{i,j} = w_{i,j} - x_i \delta_j^N$
  - where x_i is the value going out of node M_i



Alternative activation functions







- There are alternative activation functions
- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $ReLU(x) = \max(x, 0)$
- ReLU is the preferred method in hidden layers in deep networks

Today

- Neural networks
- Language models
- Word embeddings
- Word2vec

Language model

Probabilistic Language Models

- Goal: Ascribe probabilities to word sequences.
- Motivation:
  - Translation:
    - P(she is a tall woman) > P(she is a high woman)
    - P(she has a high position) > P(she has a tall position)
  - Spelling correction:
    - P(She met the prefect.) > P(She met the perfect.)
  - Speech recognition:
    - P(I saw a van) > P(eyes awe of an)

Probabilistic Language Models

- Goal: Ascribe probabilities to word sequences.
  - $P(w_1, w_2, w_3, \ldots, w_n)$
- Related: the probability of the next word
  - $P(w_n \mid w_1, w_2, w_3, \ldots, w_{n-1})$
- A model which does either is called a Language Model, LM
  - Comment: The term is somewhat misleading
    - (Probably origin from speech recognition)

Chain rule

- The two definitions are related by the chain rule for probability:
- $P(w_1, w_2, w_3, \ldots, w_n) =$
- $P(w_1) \times P(w_2 | w_1) \times P(w_3 | w_1, w_2) \times \cdots \times P(w_n | w_1, w_2, \ldots, w_{n-1}) =$
- $\prod_i^n P(w_i | w_1, w_2, \ldots, w_{i-1}) = \prod_i^n P(w_i | w_1^{i-1})$

- *P("its water is so transparent") =*

  P(its) × P(water|its) × P(is|its water)

  × P(so|its water is) × P(transparent|its water is so)

- But this does not work for long sequences
  - (we may not even have seen before)

Markov assumption

- A word depends only on the immediate preceding word
- $P(w_1, w_2, w_3, \dots, w_n) \approx$
- $P(w_1) \times P(w_2 | w_1) \times P(w_3 | w_2) \times \cdots \times P(w_n | w_{n-1}) =$
- $\prod_i^n P(w_i | w_{i-1})$

- P("its water is so transparent") ≈

  P(its) × P(water | its) × P(is | water) × P(so | is) × P(transparent | so)

- This is called a bigram model

Estimating bigram probabilities

- The probabilities can be estimated by counting
- This yields maximum likelihood probabilities
  - (=maximum probable on the training data)
- $\hat{P}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)}{count(w_{i-1})}$

Example from J&M

$$\hat{P}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I}\,|\,\texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam}\,|\,\texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am}\,|\,\texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>}\,|\,\texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam}\,|\,\texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do}\,|\,\texttt{I}) = \frac{1}{3} = .33$

General ngram models

- A word depends only on the k many immediately preceding words
- $P(w_1, w_2, w_3, \ldots, w_n) \approx$
- $\prod_i^n P(w_i | \, w_{i-k}, w_{i+1-k}, \ldots, w_{i-1}) = \prod_i^n P\left(w_i | \, w_{i-k}^{i-1}\right)$

- This is called a
  - unigram model – no preceding words
  - trigram model – two preceding words
  - *k*-gram model – *k*-1 preceding words

- We can train them similarly to the bigram model.
- Have to be more careful with the smoothing for larger *k*-s.

Generating with n-grams

- Goal: Generate a sequence of words
- Unigram:
  - Choose the first word according to how probable it is
  - Choose the second word according to how probable it is, etc.
  - = the generative model for multinomial NB text classification
- Bigram
  - Select word k according to $\hat{P}(w_i|w_{i-1})$
- k-gram
  - Select word w_i according to how probable it is given the $k-1$ preceding words $P\left(w_i|\, w_{i-k}^{i-1}\right)$

Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Unknown words

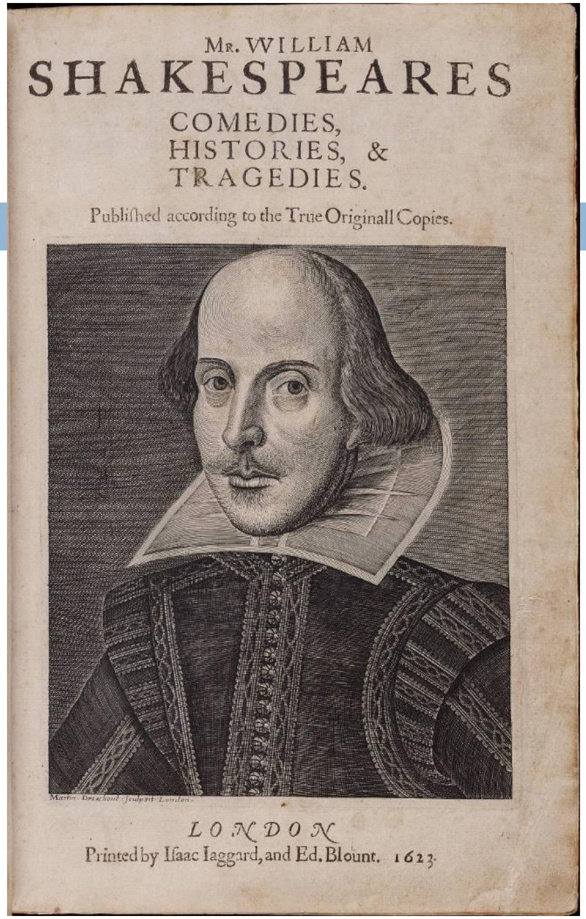
- There might be words that is never observed during training.
- Use a special symbol for unseen words during application, e.g. UNK
- Set aside a probability for seeing a new word
  - This may be estimated from a held-out corpus
- Adjust
  - the probabilities for the other words in a unigram model accordingly
  - the conditional probabilities of the *k-gram model*

Smoothing, Laplace, Lidstone

- Since we might not have seen all possibilities in training data, we might use Lidstone or, more generally, Laplace smoothing
- $\hat{P}(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)+k}{count(w_{i-1})+k\ |V|}$
  - where $|V|$ is the size of the vocabulary V.

But:

- Shakespeare produced
  - N = 884,647 word tokens
  - V = 29,066 word types
- Bigrams:
  - Possibilities:
    - $V^2 = 844{,}000{,}000$
  - Shakespeare,
    - bigram tokens: 884,647
    - bigram types: 300,000



- Add-k smoothing is not appropriate

Smoothing n-grams

Backoff

- If you have good evidence, use the trigram model,
- If not, use the bigram model,
- or even the unigram model

Interpolation

- Combine the models

Use either of this. According to J&M interpolation works better

Interpolation

- Simple interpolation:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+ \lambda_2 P(w_n|w_{n-1}) \\ &+ \lambda_3 P(w_n)\end{aligned}$$

- The λ-s can be tuned on a held out corpus
- A more elaborate model will condition the λ-s on the context
  - (Brings in elements of backoff)

Evaluation of n-gram models

- Extrinsic evaluation:
  - To compare two LMs, see how well they are doing in an application, e.g. translation, speech recognition
- Intrinsic evaluation:
  - Use a held out-corpus and measure $P(w_1, w_2, w_3, \ldots, w_n)^{\frac{1}{n}}$
    - The n-root compensate for different lengths
  - $\prod_i^n P\left(w_i \mid w_{i-k}^{i-1}\right)^{\frac{1}{n}}$ for a k-gram model
  - It is normal to use the inverse of this, called the perplexity
  - $PP(w_1^n) = \frac{1}{P(w_1, w_2, w_3, \ldots, w_n)^{\frac{1}{n}}} = P(w_1, w_2, w_3, \ldots, w_n)^{-\frac{1}{n}}$

Properties of LMs

- The best smoothing is achieved with Kneser-Ney smoothing
- Short-comings of all n-gram models
  - The smoothing is not optimal
  - The context are restricted to a limited number of preceding words.

A practical advice: Use logarithms when working with n-grams

Today

- Neural networks
- Language models
- Word embeddings
- Word2vec

Word-context matrix

- Two **words** are similar in meaning if their context vectors are similar

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

So-far

- A word w can be represented by a context vector v_w where position j in the vector reflects the frequency of occurrences of w_j with w.
- Can be used for
  - studying similarities between words.
  - document similarities
- But the vectors are *sparse*
  - Long: 20-50,000
  - Many entries are 0
- Even though *car* and *automobile* get similar vectors, because both co-occur with e.g., *drive*, in the vector for *drive* there is no connection between the *car* element and the *automobile* element.

Today

- Lexical semantics
- Vector models of documents
- tf-idf weighting
- Word-context matrices
- Word embeddings with dense vectors

Dense vectors

How?

- Shorter vectors.
  - (length 50-1000)
  - ``low-dimensional'' space
- Dense (most elements are not 0)
- Intuitions:
  - Similar words should have similar vectors.
  - Words that occur in similar contexts should be similar.

Properties

- Generalize better than sparse vectors.
- Input to deep learning
  - Fewer weights (or other weights)
- Capture semantic similarities better.
- Better for sequence modelling:
  - Language models, etc.

Word embeddings

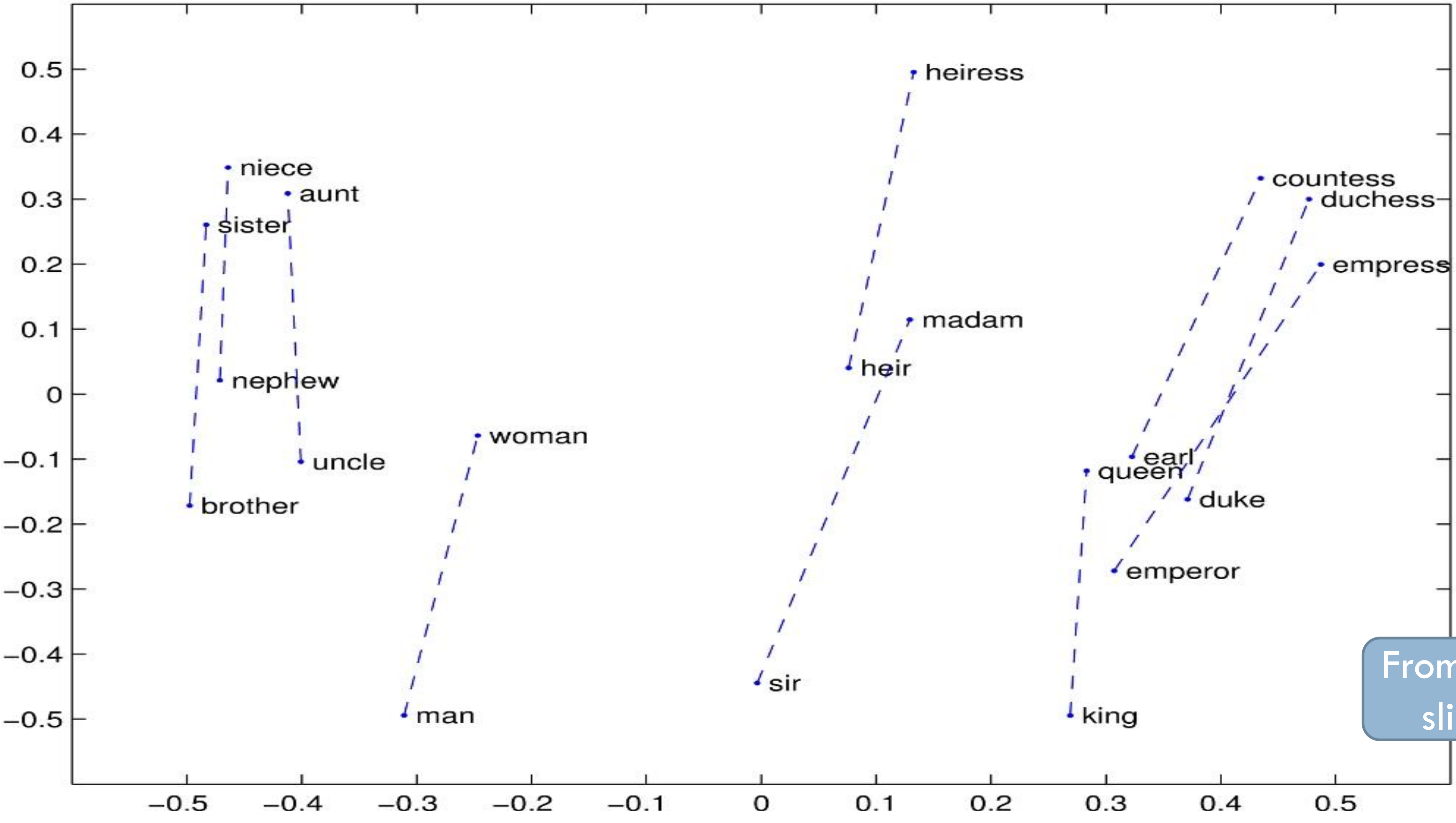
- In current LT: Each word is represented as a vector of reals
- Words are more or less similar
- A word can be similar to one word in some dimensions and other words in other dimensions

Dimensions

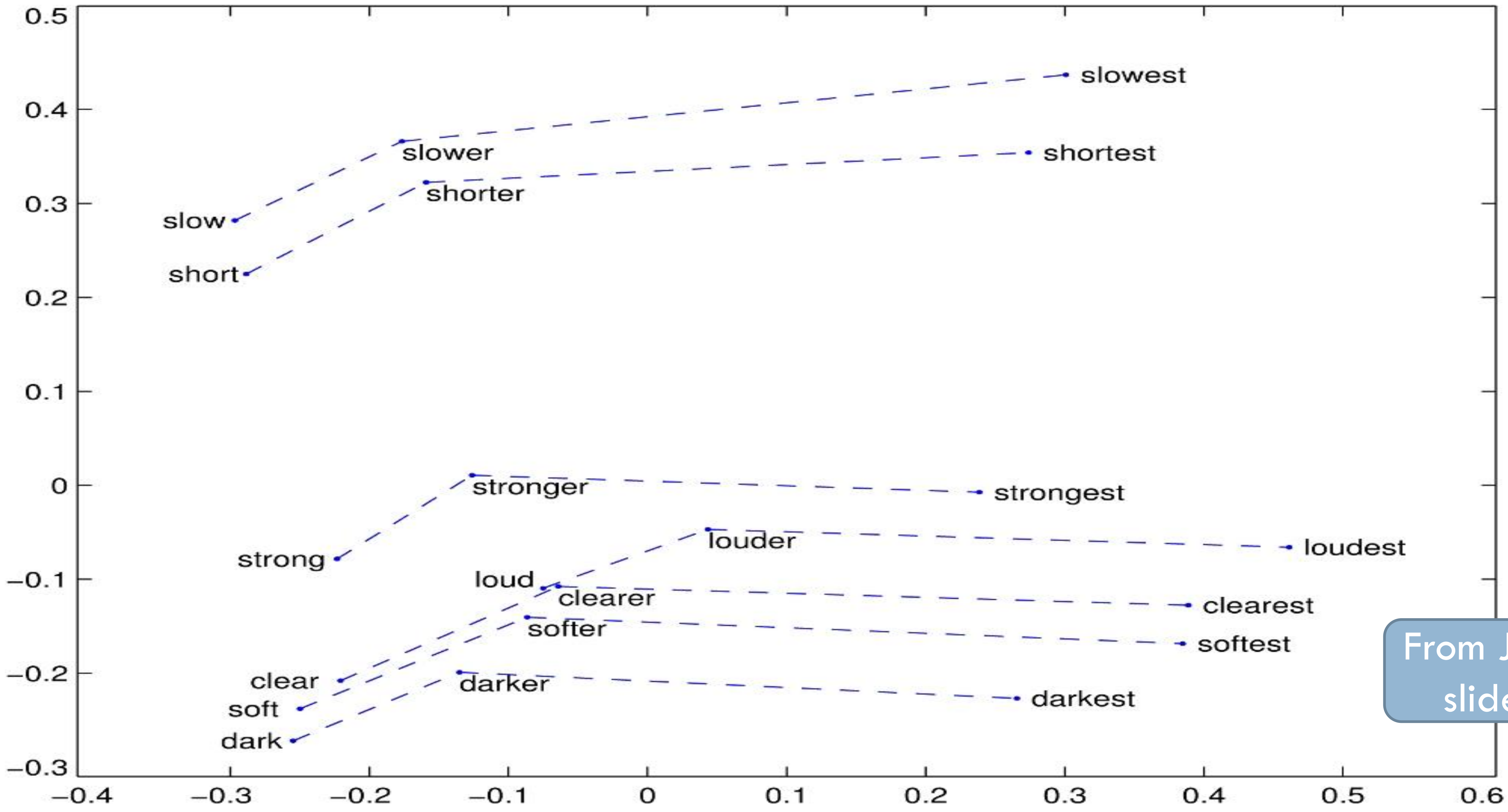
Word vectors				
dog	-0.4	0.37	0.02	-0.34
cat	-0.15	-0.02	-0.23	-0.23
lion	0.19	-0.4	0.35	-0.48
tiger	-0.08	0.31	0.56	0.07
elephant	-0.04	-0.09	0.11	-0.06
cheetah	0.27	-0.28	-0.2	-0.43
monkey	-0.02	-0.67	-0.21	-0.48
rabbit	-0.04	-0.3	-0.18	-0.47
mouse	0.09	-0.46	-0.35	-0.24
rat	0.21	-0.48	-0.56	-0.37

animal
domesticated
pet
fluffy

Figure from https://medium.com/@jayeshbahire



From J&M slides

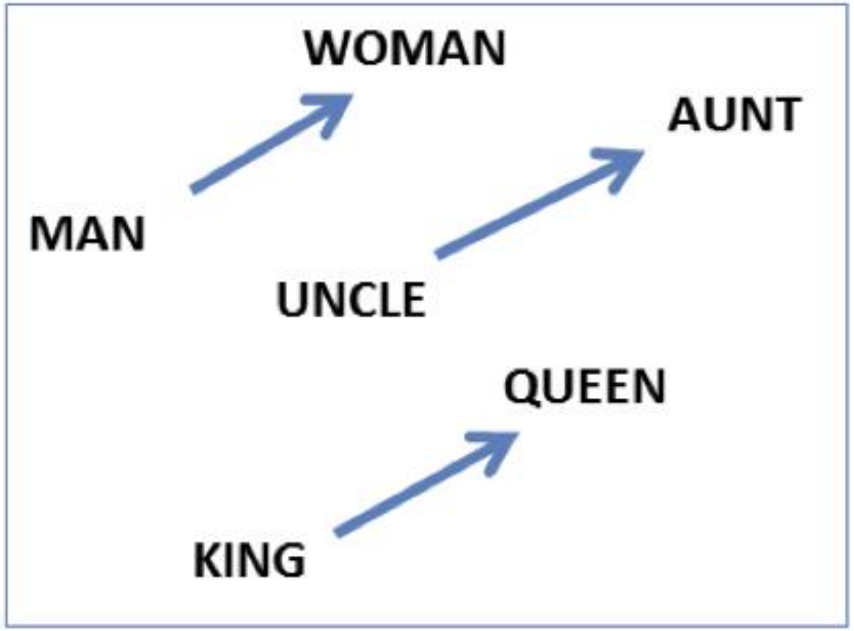


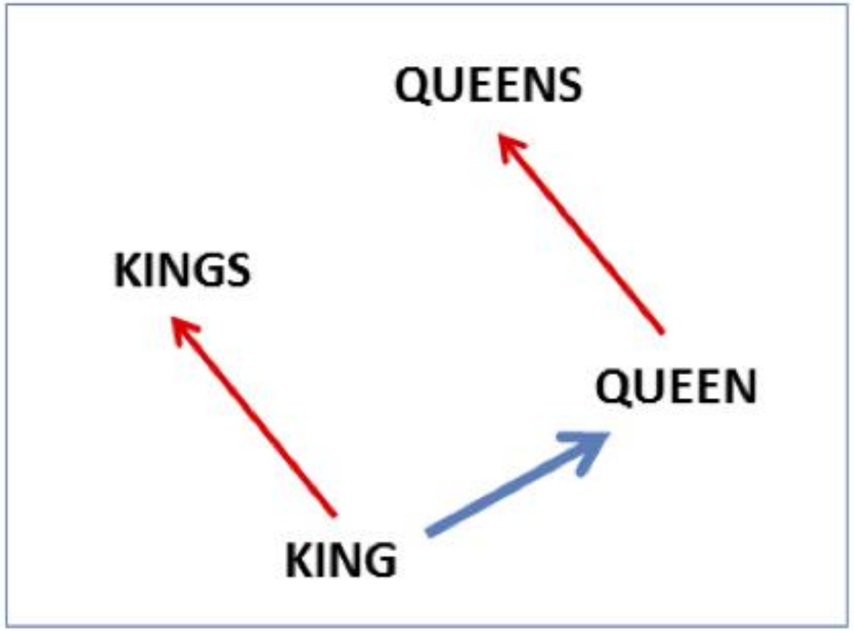
From J&M slides

Analogy: Embeddings capture relational meaning!

vector('*king*') - vector('*man*') + vector('*woman*') ≈ vector('queen')

vector('*Paris*') - vector('*France*') + vector('*Italy*') ≈ vector('Rome')



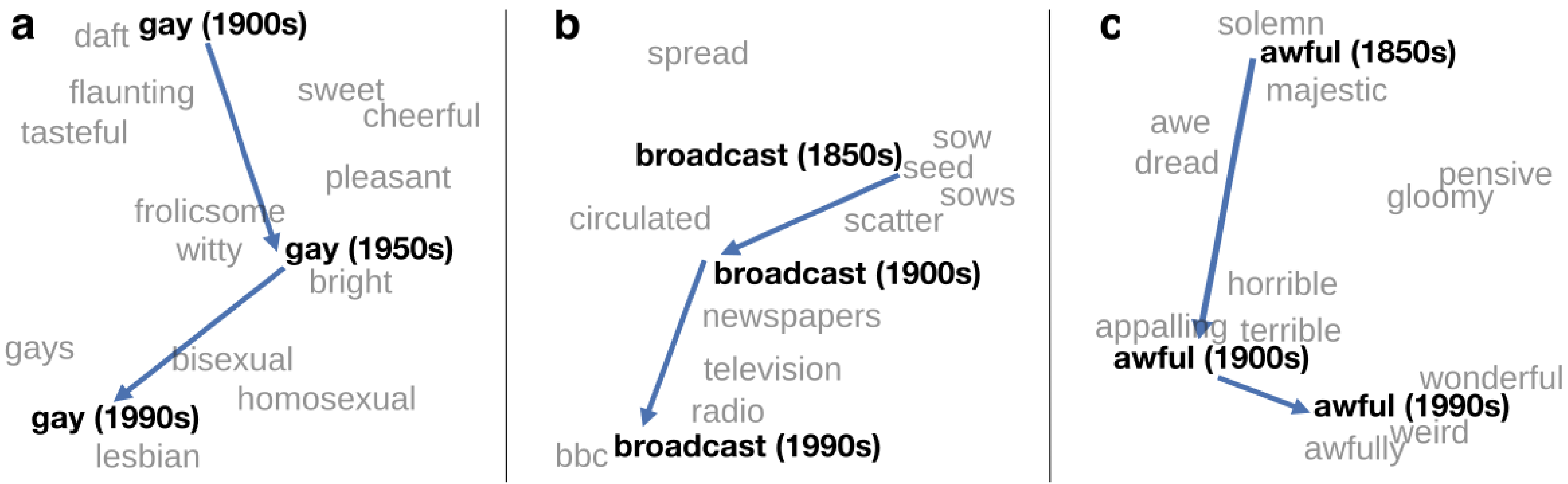


From J&M slides

Demo

- http://vectors.nlpl.eu/explore/embeddings/en/

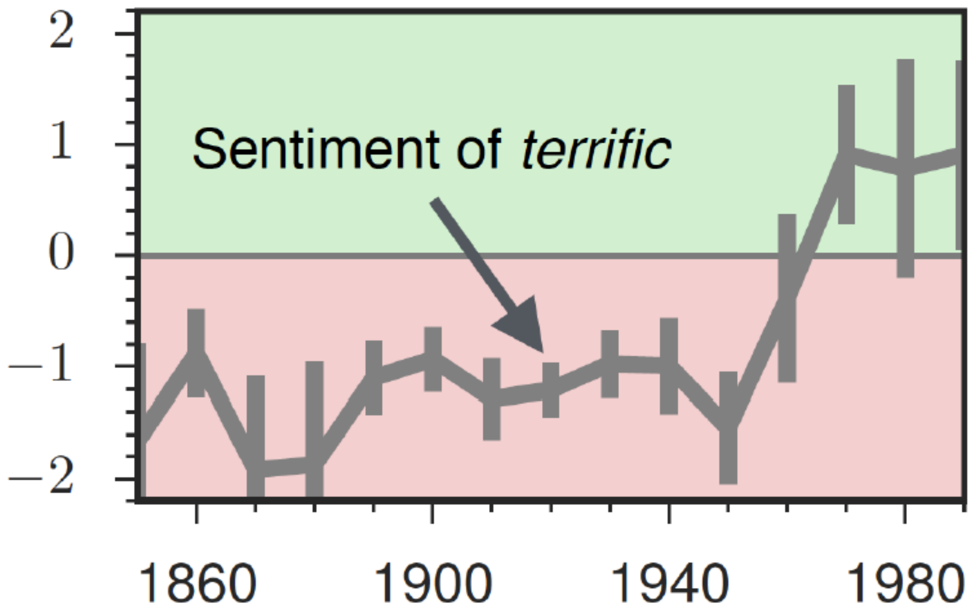
Track change of meaning of words



~30 million books, 1850-1990, Google Books data

From J&M slides

Evolution of sentiment words



- Negative words change faster than positive words

From J&M slides

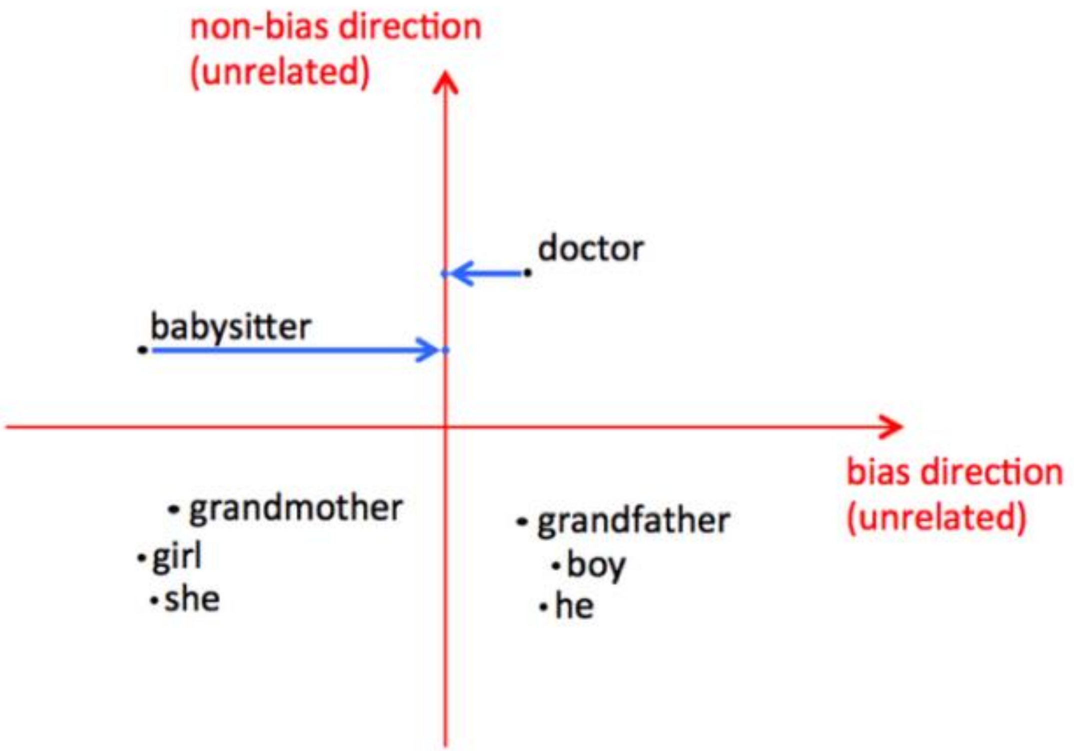
Bias

- *Man is to computer programmer as woman is to homemaker*.
- Different adjectives associated with:
  - male and female terms
  - typical black names and typical white names
- Embeddings may be used to study historical bias

Debiasing (research topic)

- Goal: neutralize the biases
- Some positive results
- But also reports that is is not fully possible

- Is debiasing a goal?
- When should we (not) debias?



https://vagdevik.wordpress.com/2018/07/08/debiasing-word-embeddings/

Evaluation of embeddings

- Extrinsic evaluation:
  - Evaluate contribution as part of an application
- Intrinsic evaluation:
  - Evaluate against a resource
- Some datasets
  - WordSim-353:
    - Broader "semantic relatedness"
  - SimLex-999:
    - Narrower: similarity
    - Manually annotated for similarity

Word1	Word2	POS	Sim-score
old	new	A	1.58
smart	intelligent	A	9.2
plane	jet	N	8.1
woman	man	N	3.33
word	dictionary	N	3.68
create	build	V	8.48
get	put	V	1.98
keep	protect	V	5.4

Part of SimLex-999

Use of embeddings

- Embeddings are used as representations for words as input in all kinds of NLP tasks using deep learning:
  - Text classification
  - Language models
  - Named-entity recognition
  - Machine translation
  - etc.

Resources

- gensim
  - Easy-to-use tool for training own models
- Word2wec
  - https://code.google.com/archive/p/word2vec/
- https://fasttext.cc/
- https://nlp.stanford.edu/projects/glove/
- http://vectors.nlpl.eu/repository/
  - Pretrained embeddings, also for Norwegian

Today

- Neural networks
- Language models
- Word embeddings
- Word2vec

Idea

- Instead of counting, use a neural network to learn a LM
- Simplest form: a bigram model:
  - For a given word w_{i-1}, try to predict the next word w_i
  - i.e. try to estimate $P(w_i | w_{i-1})$

Model

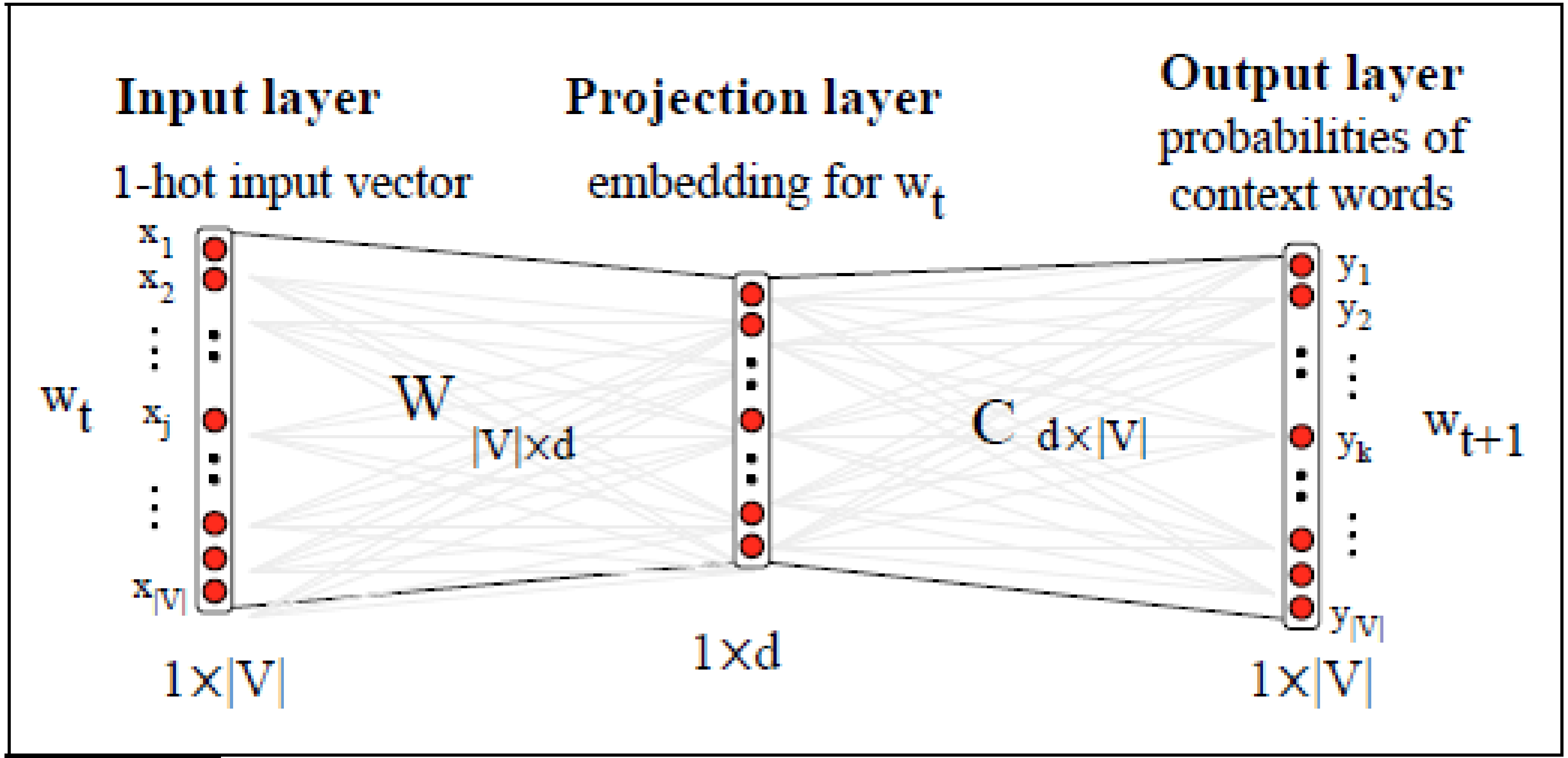


Figure 16.5 The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

From J&M 3.ed. 2018 Ch. 16

Model

- Input and output word are represented by sparse one-hot vectors
- Dim *d* typically 50-300
- Independent learning for each input word w_t:
  - Consider all possible next words for w' for this word
  - Use softmax to get a probability distribution of all next words

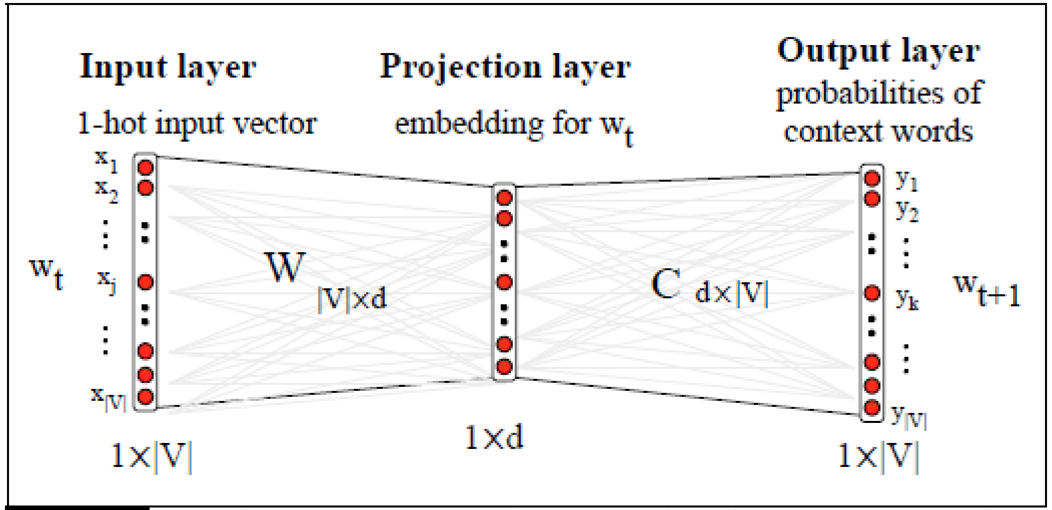


Figure 16.5 The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

Embeddings from this

- Idea: Use the weight matrix $W_{|V|\times d}$ as embeddings, i.e.:
- Represent word j by $(w_{j,1}, w_{j,2}, \dots, w_{j,d})$ = the weights that sends this word to the hidden layer
- Why? since similar words will predict more or less the same words, they will get similar embeddings

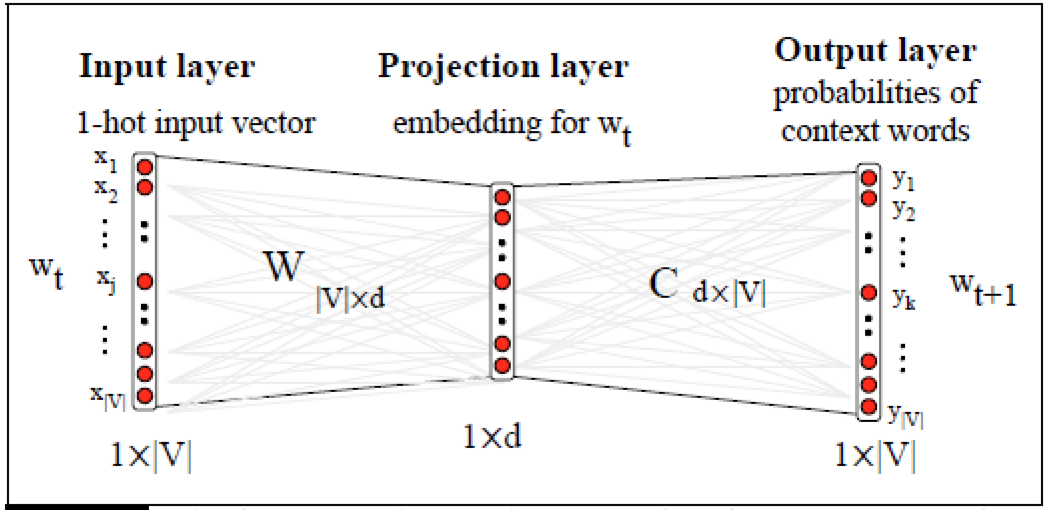


Figure 16.5 The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

Observations

- Since two words that are similar are predicted by the same words, there will also be similarities between similar words in $C_{d\times|V|}$
- This will help the training of $W_{|V|\times d}$
- We could alternatively use $C_{d\times|V|}$ as the embeddings

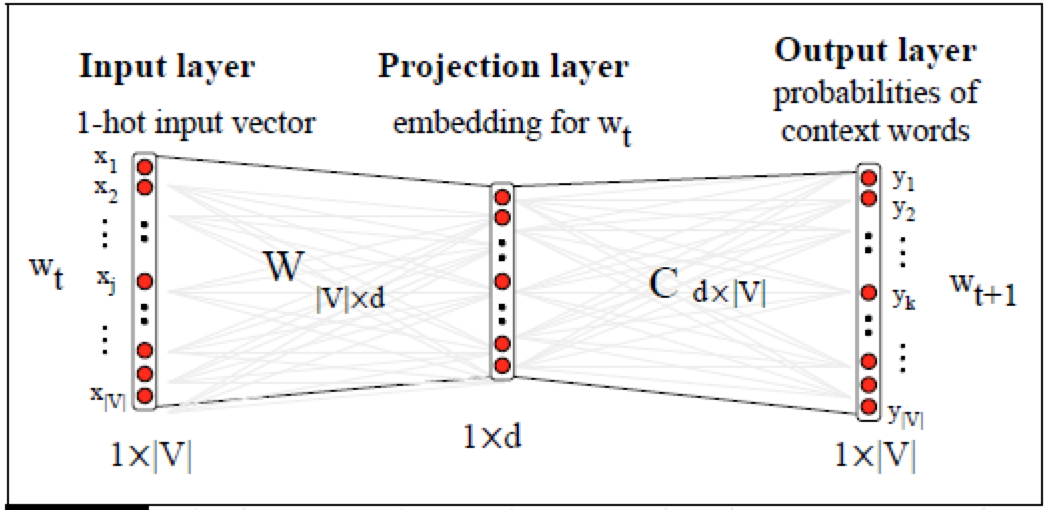
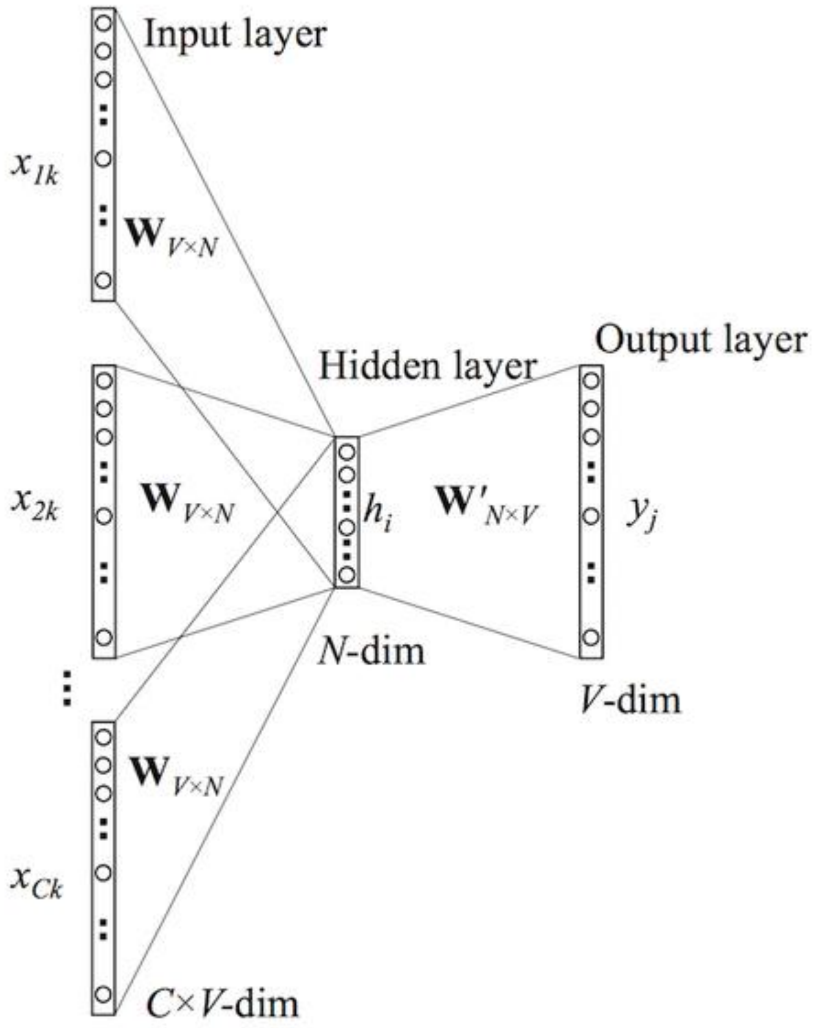


Figure 16.5 The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

CBOW

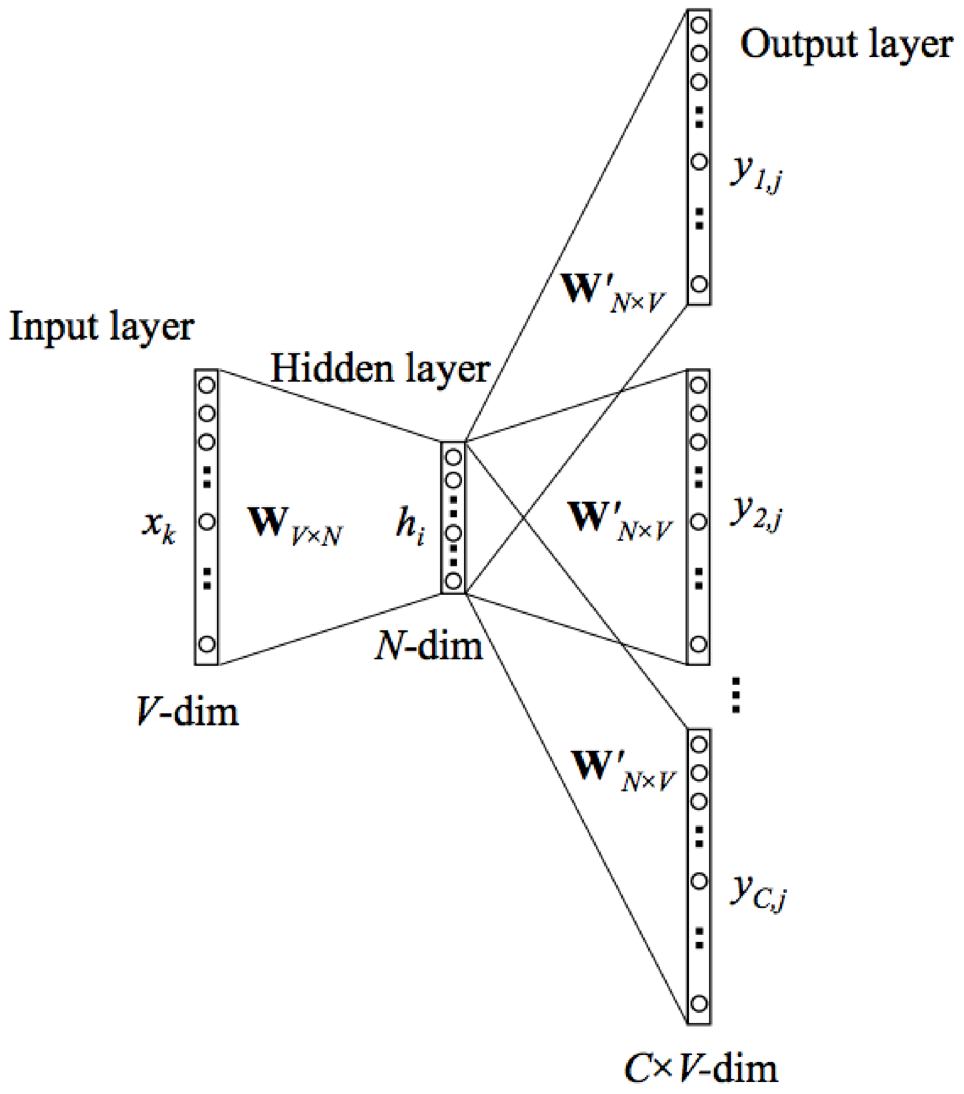
- We could generalize to predicting from a number of preceding words, e.g. 3, as indicated in the figure.
- Observe this is order-independent
- Continuous bag of words model (CBOW):
  - Predict w_t from a window $(w_{t-k}, \dots, w_{t-1,} w_{t+1}, \dots, w_{t+k})$



https://commons.wikimedia.org/wiki/File:Cbow.png

Skip-gram

- From w_t predict all the words in a window $(w_{t-k}, \ldots, w_{t-1,} w_{t+1}, \ldots, w_{t+k})$
- Assume independence of the context words, i.e. from w_t predict each of the words w in $\{w_{t-k}, \ldots, w_{t-1,} w_{t+1}, \ldots, w_{t+k}\}$
- Boils down to similar to unigram model.



https://commons.wikimedia.org/wiki/File:Skip-gram.png

Skip-gram model

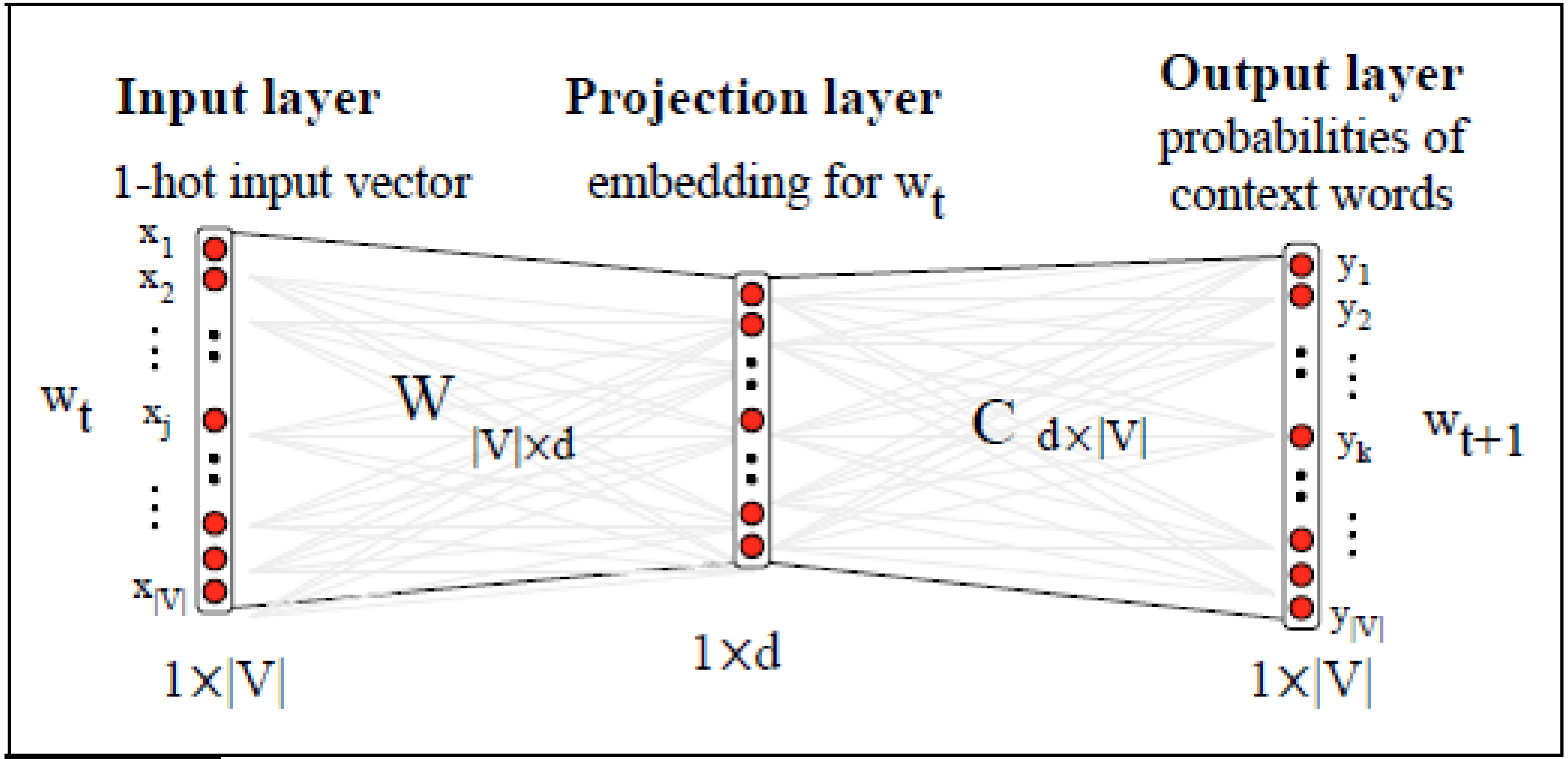


Figure 16.5 The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

From J&M 3.ed. 2018 Ch. 16

Skip-gram with negative sampling

- To train a skip gram model is expensive
- Soft-max $P(C_j|\vec{x}) = \frac{e^{\vec{w_j}\cdot\vec{x}}}{\sum_{i=1}^{k} e^{\vec{w_i}\cdot\vec{x}}}$
  - where the classes corresponds to the next word
  - i.e. in making an update for a pair (w_t, w_s) one has to calculate the weighted expression $e^{\vec{w_i}\cdot\vec{x}}$ for each word in the vocabulary
- Looking for cheaper training methods

Skip-gram with negative sampling

1. Treat the target word and a neighboring context word as a positive example.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Skip-Gram Training Data

- Training sentence:
- ... lemon, a tablespoon of apricot jam a pinch ...
- c1 c2 t c3 c4

- Training data: input/output pairs centering on *apricot*
- Asssume a +/- 2 word window

Skip-Gram Training Data

- ... lemon, a tablespoon of apricot preserves or a ...
- c1 c2 t c3 c4
- For each positive example, we'll create *k* negative examples.
  - Using *noise* words: Any random word that isn't *t*

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

How to compute p(+|t,c)?

- Word2vec
  - One of various ways to train the classifier to distinguish pos and neg words
- Intuition:
  - Words are likely to appear near similar words
  - Model similarity with dot-product!
  - Similarity $(t, c) \sim t \cdot c$
- *Problem:*
  - *Dot product is not a probability!*
    - *(Neither is cosine)*

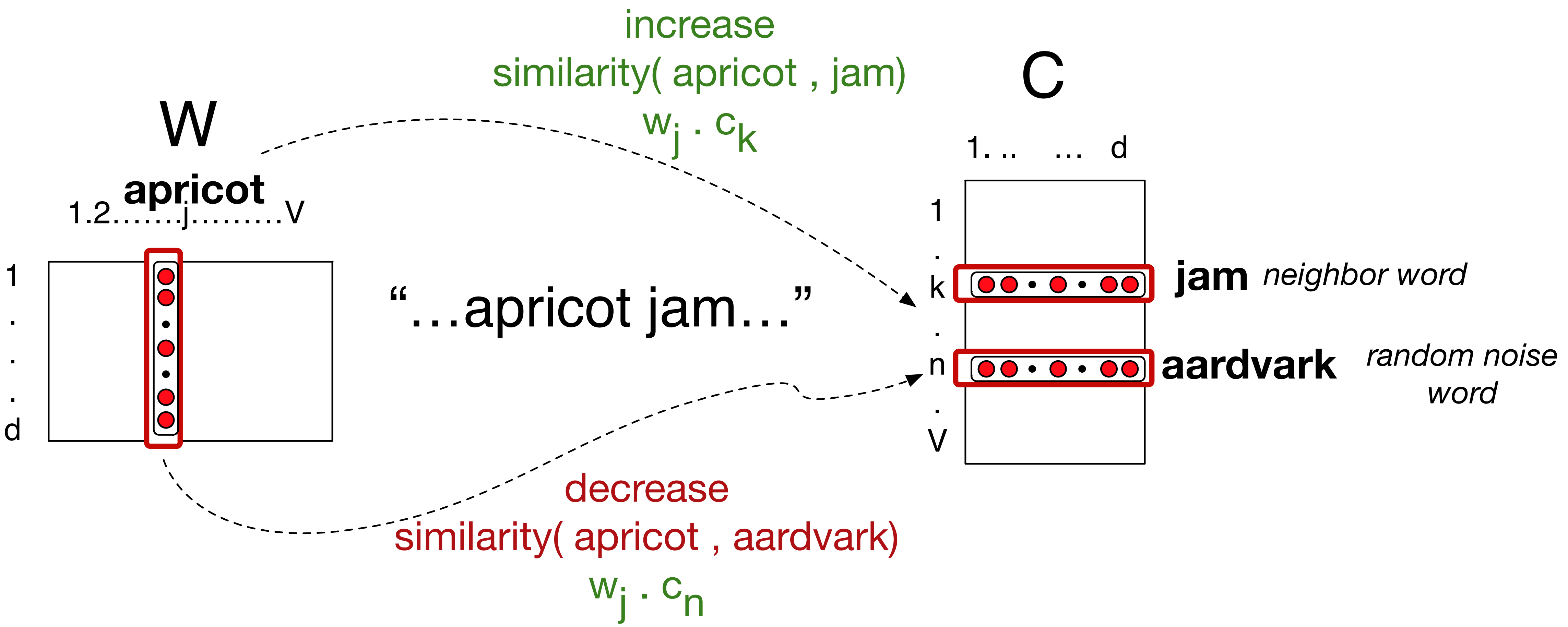
Goal

- Given a tuple (target, context)
  - (*apricot*, *jam*)
  - (*apricot*, *aardvark*)
- Calculate the probabilities
  - $P(+|t,c)$
  - $P(-|t,c) = 1 - P(+|t,c)$
- Maximize

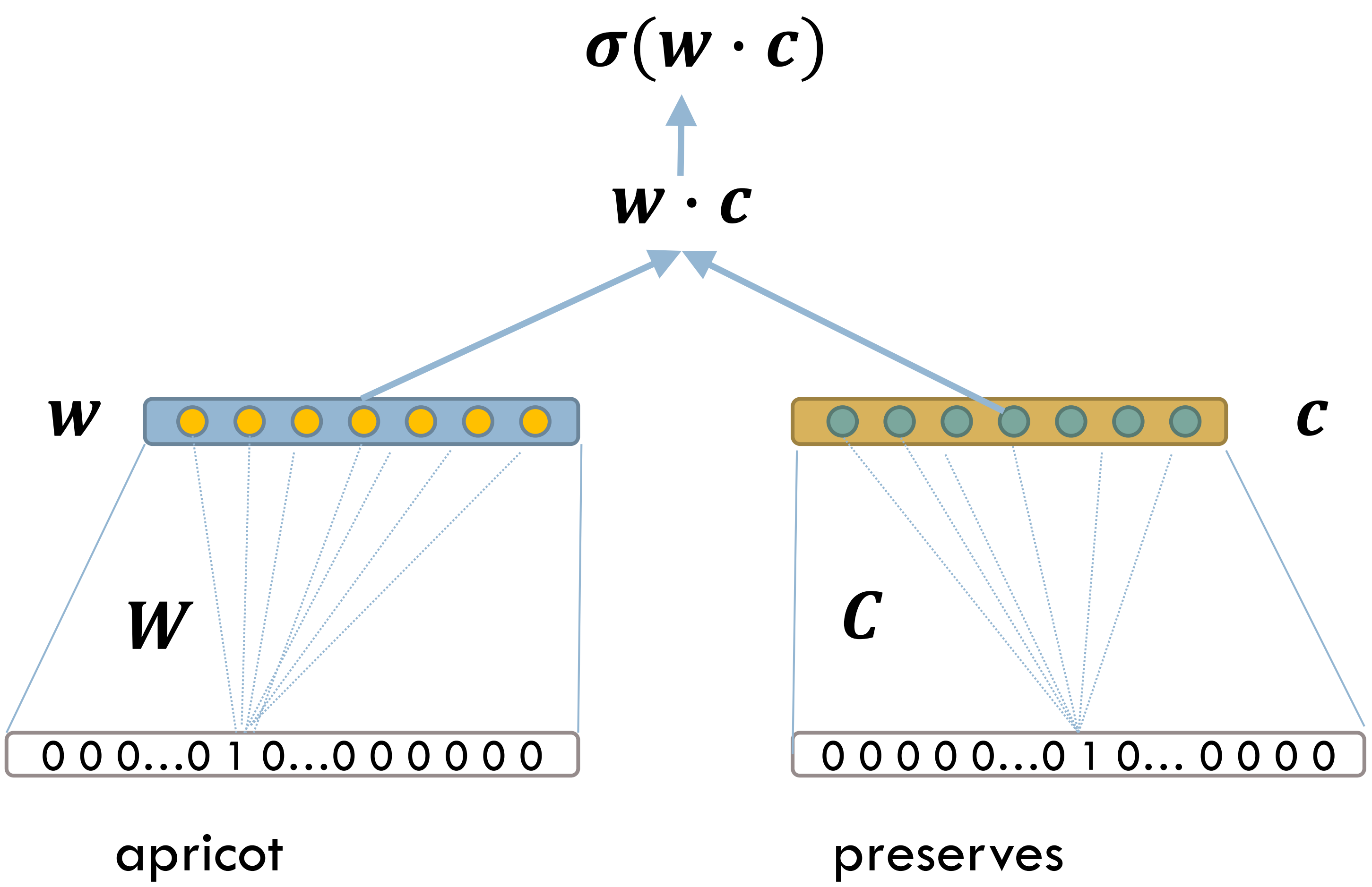
$$\sum_{(t,c)\in +} log P(+|t,c) + \sum_{(t,c)\in -} log P(-|t,c)$$

  - where

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$


increase
similarity(apricot , jam)
$w_j \cdot c_k$
W
apricot
1.2…….j………V
1
.
.
.
d
C
1. ..   …   d
1
.
k
.
n
.
V
jam
neighbor word
aardvark
random noise word
"…apricot jam…"
decrease
similarity(apricot , aardvark)
$w_j \cdot c_n$

Another view



- We feed a pair of words (w, c) to distinct hidden embedding layers
- Compare to target (1 or 0)
- Update weights
- We learn the set of embeddings W and C

Result

- We learn two separate embedding matrices W and C
- We can use W as representations for the words
  - (or combine with C in some ways)

- What have we learned:
  - If two words *w1* and *w2* occur in similar contexts
    - = with the same (or similar) context words, e.g. c,
  - then both *w1* and *w2* should have a large cosine with c,
    - hence have similar vectors.

Use of embeddings

- Embeddings are used as representations for words as input in all kinds of NLP tasks using deep learning:
  - Text classification
  - Language models
  - Named-entity recognition
  - Machine translation
  - etc.
- IN5550 Spring 2020

Resources

- gensim
  - Easy-to-use tool for training own models
- Word2wec
  - https://code.google.com/archive/p/word2vec/
- https://fasttext.cc/
- https://nlp.stanford.edu/projects/glove/
- http://vectors.nlpl.eu/explore/embeddings/en/
  - Pretrained embeddings, also for Norwegian